Universidad
Carlos III de Madrid
www.uc3m.es

Universidad Carlos III de Madrid

Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

# DESARROLLO DE APLICACIONES PARA SMART TV USANDO LA TECNOLOGÍA ANDROID TV

**Autor:** Adrián Sánchez Márquez

**Tutora:** María Celeste Campo Vázquez

Marzo 2015

Universidad Carlos III de Madrid
www.uc3m.es

Trabajo Fin de Grado

# DESARROLLO DE APLICACIONES PARA SMART TV USANDO LA TECNOLOGÍA ANDROID TV.

**Autor**

Adrián Sánchez Márquez

**Tutora**

María Celeste Campo Vázquez

Lectura del Trabajo Fin de Grado el día 9 de Marzo de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, el tribunal:

PRESIDENTA: Eva Rajo Iglesias

SECRETARIO: Rubén Cuevas Rumín

VOCAL: Patricia Arias Cabarcos

SUPLENTE: Carmen Peláez Moreno

Leganés, 24 de Febrero de 2015

# Resumen

Este documento presenta el Trabajo de Fin de Grado basado en el desarrollo de aplicaciones para Smart TV, mediante el uso de la tecnología Android TV. Se realiza un estudio y documentación de las diferentes alternativas actuales en el mundo de las Smart TV y, posteriormente, se centra dicho estudio en Android TV, la nueva apuesta de Google tras el cierre de su tecnología anterior, Google TV.

Para ello se ha documentado dicha tecnología tan reciente, presentada en el año 2014, que ha permitido conocer sus funcionalidades y capacidades para descubrir si puede tratarse de una buena alternativa a las plataformas existentes.

Para completar el estudio se ha desarrollado una aplicación para Android TV, partiendo de un código base proporcionado por Google, orientada por completo para su uso en un televisor y poder comprobar así, en primera persona, el funcionamiento de dicha tecnología.

# Contenido

# Capítulo 1: Introducción.

En este primer capítulo se presentan las diferentes motivaciones que han llevado a realizar este trabajo, los objetivos pretendidos y el plan básico de trabajo que se ha llevado a cabo para su realización.

## *1.1 Motivación del proyecto.*

Muchos años han pasado desde el comienzo del desarrollo para la transmisión de señales audiovisuales a distancia. Los primeros ensayos de televisión se iniciaron de forma paralela en Estados Unidos y Europa en los años 30 del siglo XX. Ya en la década de los 50, la televisión llegaría a los principales países del mundo. En un principio, sólo se retransmitía durante ciertas horas al día y todos los contenidos eran en directo [1].

Desde entonces y hasta hoy en día, la televisión en sí y el concepto que tenemos de ella han sufrido una transformación completa. El último gran cambio vino con la implantación de la Televisión Digital Terrestre (TDT), que en el caso de España, comenzó sobre el año 2000 y se ha ido asentando por completo durante los años posteriores hasta el cese de la emisión analógica el 3 de abril de 2010 [2]. Los cambios más representativos que trajo la llegada de la televisión digital fueron un gran aumento de la calidad de imagen y sonido y decenas de nuevos canales disponibles.

Actualmente, junto con todos estos cambios, en el mundo televisivo ha irrumpido la posibilidad de aumentar, todavía más, la funcionalidad de la televisión gracias a internet. Hoy en día, no es necesario esperar a un día y horas concretos para poder ver un contenido específico. Gracias al uso de internet, aparte de los canales de televisión, se abre un mundo totalmente nuevo para el espectador.

Es en este punto donde aparece el término de Smart TV o televisión inteligente, que combina el televisor tradicional junto con el acceso a internet y todo lo que ello conlleva como el acceso a nuevos contenidos en línea en tiempo real y bajo demanda o instalación de aplicaciones.

Todos estos avances, y la gran repercusión que siempre ha tenido la televisión en los usuarios desde su nacimiento, conforman en gran parte la motivación que ha llevado a realizar este trabajo. Por otro lado, el hecho de poder comenzar a desarrollar para un dispositivo para el cual no era habitual el desarrollo de aplicaciones, como es un televisor, también representa un gran atractivo.

## 1.2 Objetivos.

El proyecto está enfocado al estudio y documentación de Android TV, como alternativa Smart TV. Para ello se define una serie de objetivos principales, que se muestran a continuación.

- Investigar y documentar las principales alternativas de Smart TV existentes en el mercado actualmente.
- Investigar y documentar una tecnología tan reciente como es Android TV, basada en el sistema operativo Android, que tan extendido está en otro tipo de dispositivos. Estudiar las principales funcionalidades y posibilidades que ofrece Android TV.
- Desarrollar una sencilla aplicación orientada por completo para su visionado y uso en un televisor utilizando las nuevas funciones y librerías del último API de Android, pensadas específicamente para el desarrollo en televisiones.
- Realizar un plan de pruebas para comprobar el correcto funcionamiento de la aplicación y redactar unos resultados finales en consecuencia.
- Exponer conclusiones finales y proponer mejoras o pasos futuros para la aplicación.

## 1.3 Plan de trabajo.

En esta sección se muestran las diferentes tareas llevadas a cabo durante la realización del proyecto.

A continuación se muestran organizadas en diferentes apartados según la etapa de desarrollo.

1. Estudio y documentación:

   Se estudia y documenta la tecnología a utilizar, Android TV. Se realiza un estudio de una primera API (Android L Preview) y posteriormente dicho estudio se traslada al API definitiva de Android 5.0 Lollipop.

   Se investigan las diferentes funcionalidades disponibles para el desarrollo de aplicaciones en un entorno como es un televisor.

2. Instalación y configuración:

   Se realiza un pequeño estudio para elegir el entorno a utilizar para el desarrollo y finalmente resulta ser *Android Studio*.

   Instalación y documentación de dicho entorno junto con la configuración, tanto del proyecto orientado a TV, como del emulador Android donde se llevarán a cabo las pruebas de la aplicación.

3. Diseño y desarrollo de una aplicación para Android TV:

   Desarrollo del código de la aplicación para Android TV, a partir de código oficial proporcionado por Google. Diseño de aspectos gráficos y obtención de recursos en línea para su uso en la aplicación.

4. Pruebas de la aplicación:

   Se comprueba el correcto funcionamiento de la aplicación en el entorno de emulación y se consigue probar en un dispositivo físico prestado.

5. Redacción y corrección de la memoria.

En la ilustración 1 se muestra el diagrama de Gantt. Se debe tener en cuenta que las fechas y duraciones de las tareas son orientativas.

## Diagrama de Gantt

| Id | Nombre de tarea | Duración | Comienzo |
|---|---|---|---|
| 1 | **1. Estudio y documentación** | **25 días** | **lun 01/09/14** |
| 2 | 1.1 Estudio API Android L Preview | 5 días | lun 01/09/14 |
| 3 | 1.2 Estudio API 21 Android 5.0 | 15 días | lun 08/09/14 |
| 4 | 1.3 Estudio tecnologías Smart TV | 5 días | lun 29/09/14 |
| 5 | **2. Instalación y coniguración** | **7 días** | **lun 06/10/14** |
| 6 | 2.1 Android SDK | 2 días | lun 06/10/14 |
| 7 | 2.2 Entorno Android Studio | 2 días | mié 08/10/14 |
| 8 | 2.3 Emulador Android TV | 3 días | vie 10/10/14 |
| 9 | **3. Diseño y desarrollo de la aplicación** | **48 días** | **mié 15/10/14** |
| 10 | 3.1 Estudio código proporcionado por Google | 20 días | mié 15/10/14 |
| 11 | 3.2 Diseños de la aplicación | 3 días | mié 12/11/14 |
| 12 | 3.3 Modificación funcionalidad existente | 10 días | lun 17/11/14 |
| 13 | 3.4 Desarrollo nuevas funcionalidades | 15 días | lun 01/12/14 |
| 14 | **4. Pruebas de la aplicación** | **4 días** | **lun 22/12/14** |
| 15 | 4.1 Pruebas en emulador | 3 días | lun 22/12/14 |
| 16 | 4.2 Pruebas en dispositivo físico | 1 día | jue 25/12/14 |
| 17 | **5. Redacción y correción de memoria final** | 45 días | vie 26/12/14 |

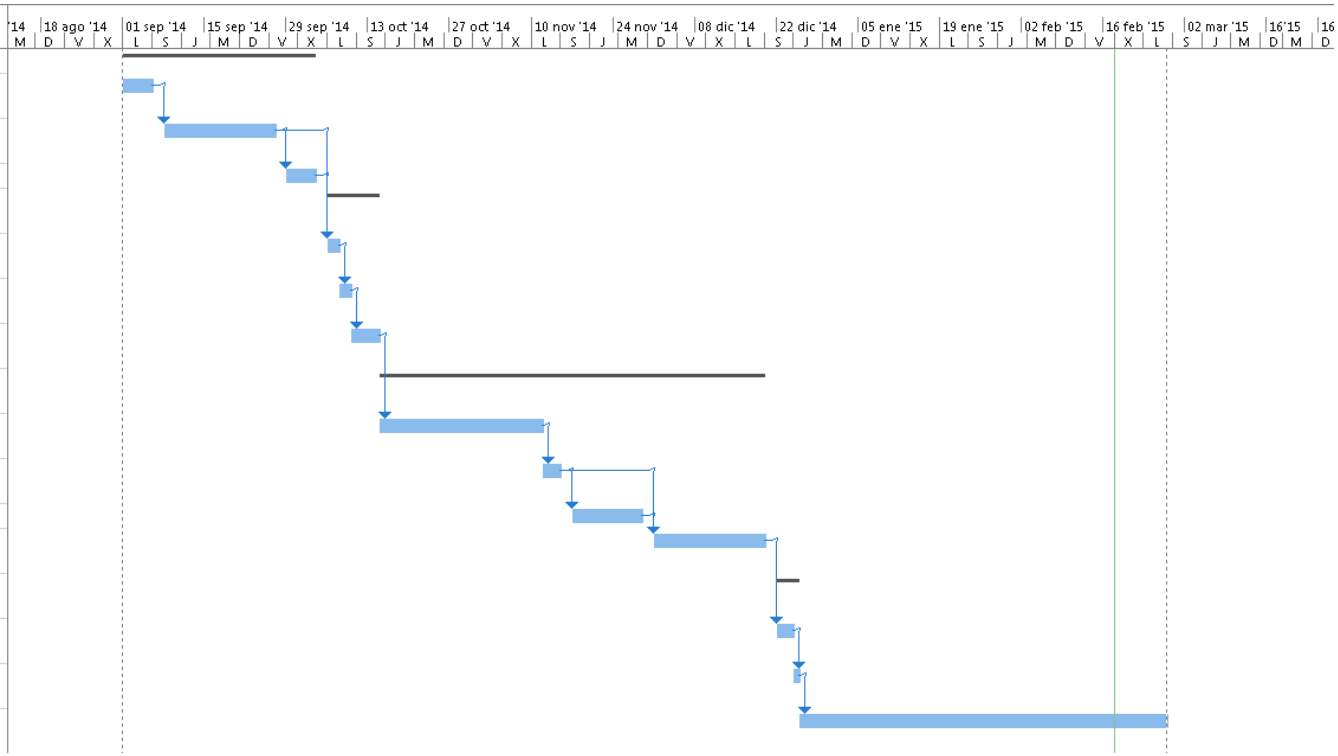


*Ilustración 1. Diagrama de Gant*

## *1.4 Presentación de la memoria.*

En esta sección se describe de manera resumida cada uno de los Capítulos de esta memoria.

En el capítulo 1 se introduce brevemente el proyecto y se comentan aspectos como las diferentes motivaciones que han llevado a realizar el proyecto, los objetivos que se pretenden alcanzar y una breve explicación del plan de trabajo llevado a cabo.

En el capítulo 2 se estudia el concepto de Smart TV y el impacto y novedades que aporta sobre la televisión tradicional. Se explican las principales plataformas que existen sobre Smart TV, incluida la tecnología Android TV, punto principal de este trabajo, y sus ventajas y aspectos más relevantes en comparación con el resto de tecnologías. Además, en este capítulo se introduce, de forma breve, los conceptos básicos y principales sobre el sistema operativo Android y los principales componentes de una aplicación desarrollada en dicho sistema.

En el capítulo 3 se detallan los requisitos necesarios para comenzar a desarrollar en Android TV, junto con una breve explicación sobre la configuración del entorno de desarrollo y primeros pasos. Además se incluye una guía explicando diferentes consideraciones a la hora de desarrollar para un televisor.

El capítulo 4 se corresponde con el diseño y desarrollo de la aplicación e incluye un pequeño resumen, estructuración de dicha aplicación y explicación de cada pantalla que la conforman, junto con una guía de desarrollo, fragmentos de código significativos y capturas de la aplicación resultante.

El capítulo 5 incluye una sección describiendo una serie de pruebas llevadas a cabo para comprobar el correcto funcionamiento de la aplicación, junto con un apartado en el que se describen las diferentes limitaciones que han surgido durante la realización del proyecto.

En el capítulo 6 se escriben las conclusiones obtenidas tras el desarrollo del trabajo y se proponen trabajos futuros y siguientes pasos que se podrían llevar a cabo para mejorar la aplicación.

Por último se incluyen dos secciones correspondientes al marco regulador y el entorno socio-económico, incluyendo un breve presupuesto.

# Capítulo 2: Estado del Arte.

En este segundo capítulo se van a describir y explicar conceptos relacionados con Smart TV, diferentes alternativas y plataformas, las primeras versiones de Google TV y su posterior apuesta por Android TV.

Además, se incluirá una breve introducción técnica sobre Android en general, necesaria para conocer ciertos conceptos sobre dicho sistema operativo y facilitar la comprensión del desarrollo de la aplicación, durante el capítulo 4.

## *2.1 Smart TV*

Con la llegada de internet y las conexiones de banda ancha, aparecieron nuevas posibilidades a la hora de ver o consumir contenidos multimedia. La televisión, como hasta entonces se conocía, se limitaba a emitir un número concreto de canales simultáneamente. Sin embargo, la conexión a internet en otros dispositivos, como los ordenadores, proporcionó a éstos nuevas funcionalidades y servicios que, en algunos casos, provocaron que el dispositivo principal a la hora de ver contenidos multimedia se "trasladara" al ordenador.

Aparecieron los términos de televisión a la carta o vídeo bajo demanda [3], que no eran posibles en un televisor. Gracias a internet, el usuario puede elegir en cualquier momento qué programa ver, sin necesidad de adaptarse al horario de emisión, al mismo tiempo que puede visitar sus redes sociales, navegar por internet o compartir archivos en red.

El término Smart TV o televisión inteligente, da nombre a aquellos televisores en los que se produce una integración de internet y de las características Web 2.0 junto con las funcionalidades de un televisor tradicional [4]. También se puede describir, de otro modo, como la convergencia entre un ordenador y un televisor, lo que proporciona un gran número de nuevas funcionalidades y servicios que un televisor puede ofrecer [5]:

- Multitud de aplicaciones orientadas para su uso en un televisor, así como juegos.
- Acceso a servicios basados en internet, mediante el protocolo *IPTV (Internet Protocol Television)* [6], búsqueda y navegación online, contenidos personalizados, redes sociales, etc.
- Compartición de contenido entre otros dispositivos o equipos conectados en la red, o visualización de los mismos mediante el servicio *DLNA (Digital Living Network Alliance)* [7], que proporciona unos estándares para la intercomunicación entre diferentes dispositivos multimedia.

- Controlar el televisor de forma remota mediante aplicaciones desarrolladas para dispositivos móviles.

Todas estas características comenzaron a aparecer en el año 2005, pero no fue hasta finales de 2010 cuando apareció el término Smart TV, para dar nombre a dicha plataforma. Cabe destacar que, en un primer momento, se utilizó el término *Internet TV* [4].

### 2.1.1 Plataformas Smart TV integradas en televisores

Existen diversas plataformas de Smart TV. Desde sus inicios, estas plataformas dependen en gran parte de cada fabricante, ya que cada uno elige y diseña su sistema operativo, aunque, por norma general todas se construyen sobre un núcleo Linux. Por ello, se trata de plataformas de software libre, pero cada fabricante tiene su propio SDK para el desarrollo de aplicaciones.

El diseño está totalmente orientado para su uso en un dispositivo como es un televisor, que no cuenta con pantalla táctil y se va a visualizar desde cierta distancia. Para ello, los elementos en pantalla están ordenados de la forma más intuitiva y sencilla posible, ya que el dispositivo de entrada más común será a través de un control remoto con flechas de dirección.

A continuación se van a describir las principales alternativas que existen en el mundo de Smart TV, desde las primeras plataformas integradas en los televisores, junto con los nuevos sistemas operativos que han sido anunciados por diferentes fabricantes y alternativas *Set-Top-Boxes (STB),* para dotar de "inteligencia" a un televisor tradicional y convertirlo en Smart TV [8].

En cuanto a los televisores Smart TV integrados, todos los grandes fabricantes de televisores comenzaron a comercializarlos a partir del año 2005. Cada fabricante cuenta con su propia plataforma Smart TV, pero, en esencia, las funcionalidades son muy similares, la gran diferencia radica en la interfaz de usuario y las aplicaciones disponibles. Cada plataforma cuenta con acceso a un mercado de aplicaciones desde donde descargar e instalar las aplicaciones que cada usuario considere [9] [10]. El listado de fabricantes es muy amplio, por lo que es inviable comentar todos ellos.

A continuación describiremos brevemente las plataformas disponibles más destacadas.

### *Samsung Smart TV*

Samsung es una de las primeras compañías que comenzó a comercializar televisores Smart TV en el año 2007. Las primeras versiones vienen con una versión modificada del sistema operativo Android HoneyComb y permite la instalación de aplicaciones desde su propia plataforma, *SamsungApps* [11].

Más tarde, aparecería una nueva versión de esta plataforma que incluye nuevas funcionalidades como la visualización en multi pantalla, control por voz y gestos, nuevo control remoto y ampliación del mercado de aplicaciones disponibles [12].

### *LG SmartTV*

La gran apuesta de LG, también el año 2007, fue su plataforma denominada *NetCast,* que permite una navegación sencilla en el televisor, además de todas las características propias de un Smart TV y la instalación de aplicaciones desde su propio servicio *LG Apps* [13] [14].

### *Google TV*

En el año 2010, Google anunció su propia plataforma para Smart TV, denominada Google TV, desarrollada junto con Sony, Intel y Logitech. A diferencia de otras plataformas comentadas anteriormente, cuyos sistemas se basaban en Linux, Google pretende integrar su sistema operativo para dispositivos móviles, Android, en los televisores, de forma nativa, adaptado a un televisor. Se lanzó con la versión Android 3.1 HoneyComb.

Con ello, se abría un nuevo horizonte en el mundo de los Smart TV. El sistema operativo Android está extendido por todo el mundo en millones de dispositivos, desde smartphones y tablets, pasando por, incluso, relojes de pulsera. Una de las grandes ventajas es, que al estar tan extendido y tener tal aceptación por parte de los usuarios, la cantidad de aplicaciones y desarrolladores para este sistema es muy superior a los demás competidores. Gracias a Google TV, se abrió la posibilidad de instalar las mismas aplicaciones que en los smartphones o tablets, simplemente accediendo al inmenso mercado de aplicaciones Android, *Play Store.* Las demás plataformas tienen su propio *market* de aplicaciones cuyo contenido viene definido por el fabricante, y es mucho más reducido [15]. Con ello, el número de aplicaciones es casi infinito y, de hecho, es muy sencillo adaptar cualquier aplicación Android para su uso en un televisor.

Google TV salió a la venta como *set-top-box* y también integrado en un reducido número de modelos de televisores, Sony en su gran mayoría. Sin embargo, la aventura de Google TV no duró mucho. Gran parte de ello fue debido a la falta de actualización del sistema, que quedó estancado en una versión de Android muy desfasada. A pesar de la expectación que suscitó en los usuarios, Google TV podría decirse que pasó "de

puntillas", sin apenas información y muy poca publicidad por parte de Google. Apenas se comercializó y daba la sensación de ser un proyecto "olvidado" por parte de la compañía [16].

Como consecuencia, tal y como se presagiaba, Google anunció el cierre de Google TV a principios del año 2014. Sin embargo, en su lugar se anunció **Android TV,** una nueva apuesta con la que se pretende integrar por completo un dispositivo como es el televisor, a la plataforma Android existente en otros dispositivos como *smartphones*, *tablets*, automóviles y relojes de pulsera (*Android Wear*).

Sin embargo, recientemente se han anunciado nuevos sistemas operativos para plataformas Smart TV. Varios fabricantes han decidido optar por sistemas más modernos, a la par que se diferencian más de sus competidores.

### *Smart TV basado en Tizen OS*

Samsung ha anunciado que sus nuevos televisores utilizarán el sistema operativo Tizen OS, ya presente en algunos modelos de smartphones y smartwatches. Se trata, igual que Android, de un sistema de código abierto y está pensado para facilitar la navegación por parte del usuario, gracias a su nuevo diseño [17] [18].

"*Tizen para televisión inteligente ofrece una pila de Linux completa, abierta y basada en estándares, optimizada para los dispositivos de sala de estar, como reproductores de Blu-ray, decodificadores y televisores digitales. Está diseñado para ofrecer una experiencia de televisión conectada a internet, permitiendo a los usuarios disfrutar de acceso a múltiples aplicaciones, servicios y multimedia personales, todo mientras ve la televisión*" [19].

### *Smart TV basado en WebOS*

Igual que Samsung, LG ha cambiado la plataforma que utilizan sus televisores Smart TV. En este caso, LG ha optado por WebOS 2.0, mucho más orientado a contenidos de alta definición y con un diseño simple y mucho más sencillo de utilizar. Durante el año 2014 aparecieron los primeros modelos y durante el año 2015, todos los televisores de esta compañía utilizarán WebOS como sistema operativo [20].

Se trata de un sistema basado en "cartas" o paneles para facilitar la multitarea. Está muy orientado a su utilización a base de gestos y con el teléfono móvil como control remoto.

### *Smart TV basado en Firefox OS*

Recientemente se ha anunciado que el sistema operativo de la compañía Mozilla, Firefox OS estará presente en los televisores del fabricante Panasonic durante el año 2015.

Firefox OS se trata de un sistema operativo basado en la potencia de HTML5 y, en el caso de los televisores, promete una navegación muy sencilla, mercado de aplicaciones, conexión con múltiples dispositivos y servicio de notificaciones en pantalla [21].

## 2.1.2 Plataformas Smart TV *Set-top-boxes (STP)*

Hasta este punto, se ha hablado de televisores inteligentes, Smart TV, y sus bondades con respecto al televisor tradicional. Como alternativa a la compra de un Smart TV, existe la posibilidad de "dotar" de las mismas funcionalidades a un televisor convencional [22].

Esto se consigue gracias a los **Set-top-Boxes** o **STP,** periféricos con las capacidades de un Smart TV que se conectan a un televisor para visualizar los contenidos. Existen muchas alternativas, las más importantes se explican a continuación.

### *Apple TV*

Se trata de la apuesta para televisores de la conocida compañía Apple. Se basa en el sistema operativo propia de dicha compañía, iOS, y tiene la gran desventaja de ser compatible únicamente con dispositivos con el mismo sistema, mediante el servicio *AirPlay.* Como punto a favor, cuenta con acceso a la gran plataforma *iTunes,* donde se puede adquirir todo tipo de contenido multimedia, como música y películas [23].


*Ilustración 2. Interfaz principal Apple TV. Fuente: appleadictos.com*

Como se aprecia en la ilustración 2, la interfaz cuenta con una lista de contenido destacado en la parte superior, seguido de 4 opciones: la primera, películas, dirige al menú de *iTunes Movie Store,* para adquirir últimos estrenos o las películas mejor valoradas. La opción de Música sincroniza la música que el usuario tiene en su cuenta de *iTunes* y poder escucharlas desde la nube. Por último, la opción Ordenadores y Ajustes. La primera sirve para enlazar un ordenador Mac o un PC con *iTunes* sin necesidad de cables.

La gran baza de Apple TV, como se ha comentado anteriormente, es el protocolo inalámbrico *AirPlay*. Permite la conexión inalámbrica de dispositivos como iPhone e iPad, para ver el contenido de estos en pantalla grande. La gran mayoría de aplicaciones y juegos son compatibles, pero la gran limitación es la baja compatibilidad con dispositivos, sólo productos Apple [24].

### *Amazon Fire TV*

La alternativa de la compañía Amazon. En este caso, Amazon Fire TV cuenta con un hardware mucho más potente, que permite la ejecución de aplicaciones o juegos sin problemas.

La interfaz es muy intuitiva, con acceso a todo tipo de contenidos, películas, programas de televisión, aplicaciones, juegos, fotos, etc. La visualización del contenido es de tipo "cartas" o paneles y es compatible con cualquier dispositivo móvil Android o iOS, que pueden actuar como control remoto instalando una sencilla aplicación.

La gran baza de Amazon Fire TV es su amplio catálogo, que, aparte de acceder a un gran catálogo de aplicaciones, gracias al market de Amazon, cuenta con un número de juegos muy elevado. De hecho, uno de los periféricos compatibles con Amazon Fire TV, se trata de un mando pensado exclusivamente para juegos [25].

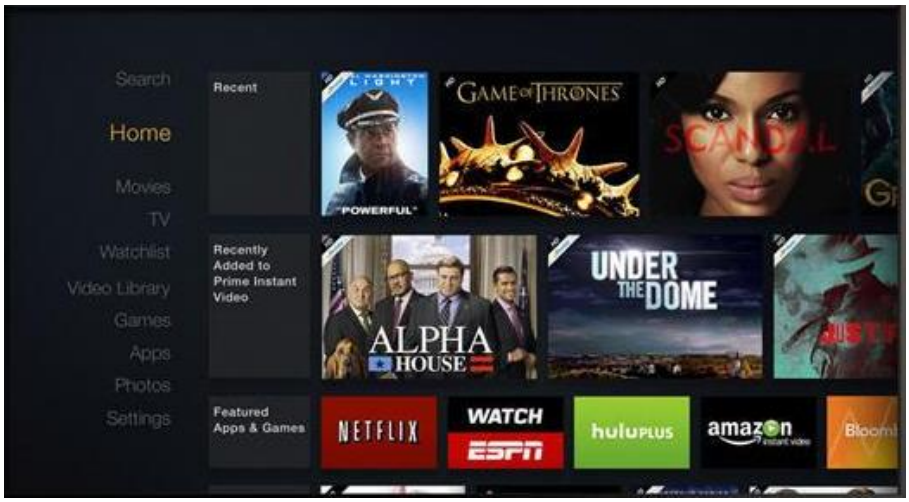


*Ilustración 3. Interfaz Amazon Fire TV. Fuente: amazon.com*

### *Roku 3*

La compañía Roku es especialista en set-top-boxes y entre sus dispositivos se puede destacar el Roku 3. Es un dispositivo perfecto en cuanto a streaming de vídeo. Cuenta con numerosos canales, con actualizaciones continuas. Sin embargo, no está orientado a la reproducción de contenido local, que pertenezca al usuario, y para ello es necesario la instalación de alguna aplicación de terceros.

En cuanto aplicaciones, existe un gran número de ellas, pensadas sobre todo para la recepción de vídeo por internet. Una gran característica es su motor de búsqueda, que realiza búsquedas en infinidad de servicios diferentes de forma simultánea.

Igual que Amazon Fire TV, es compatible con cualquier dispositivo móvil como control remoto. En un principio, la duplicación de pantalla mediante la conexión con dispositivos móviles no era posible. Sin embargo, ya existen aplicaciones compatibles con esta función, como Netflix o Youtube, y poco a poco se está extendiendo a otras aplicaciones [26].

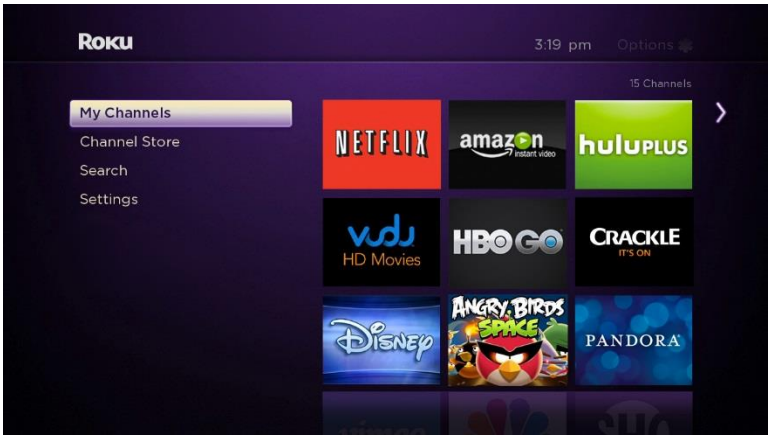


*Ilustración 4. Interfaz Roku 3. Fuente: roku.com*

### *Google Chromecast*

Chromecast es un dispositivo de transmisión de contenido multimedia que se conecta al puerto HDMI del televisor. Se necesita un teléfono o tablet Android, un iPhone, un iPad, un portátil Mac o Windows o un Chromebook para enviar las aplicaciones y contenido favoritos directamente a la TV [27].

Es decir, no es una alternativa Smart TV propiamente hablando, ya que está pensado principalmente para la visualización de contenidos procedentes de otros dispositivos.

Es compatible con varias aplicaciones de streaming de vídeo, pero el catálogo es mucho más reducido que sus competidores anteriores.

Como punto muy a su favor, cabe destacar su precio, de tan sólo 35€.

*Ilustración 5. Ejemplo uso ChromeCast con tablet y smartphone. Fuente: google.es/Chrome*

### 2.1.3 Plataforma Smart TV estudiada: Android TV

Tras la decepción que supuso Google TV, la compañía centró sus esfuerzos en renovar por completo la plataforma y para ello anunció, a finales de junio de 2014, Android TV. Para no repetir errores del pasado, esta nueva plataforma utiliza la última versión Android del momento, *Android 5.0 Lollipop.*

Cuenta con todas las ventajas de Google TV en cuanto a poder tener un sistema Android nativo en el televisor, con todo lo que ello conlleva, además de contar con la última versión disponible del sistema y una interfaz totalmente renovada, más sencilla y centrada en el contenido, construida sobre un diseño de lenguaje responsivo denominado ***Material Design*** [28] [29].

Es compatible con dispositivos más allá de la compañía Google, se puede compartir contenido desde dispositivos iOS, Macs y PCs con Windows y, además incorpora un novedoso sistema de recomendaciones personalizadas para el usuario [30].

Los primeros dispositivos con este sistema comenzarán a aparecer durante el año 2015, integrado en ciertos televisores y como set-top-box, denominado ***Nexus Player***.

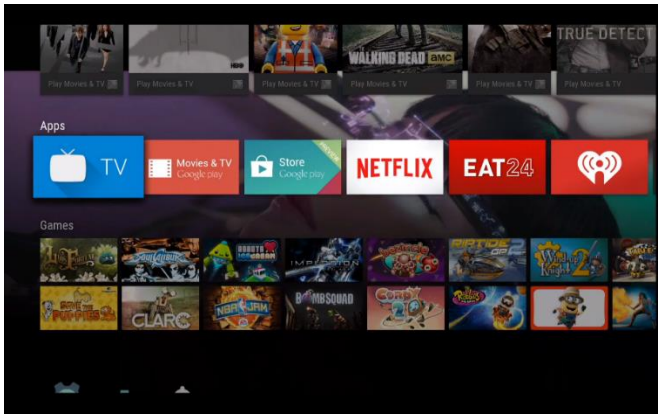


*Ilustración 6. Interfaz Android TV. Fuente: elandroidlibre.com*

A modo de resumen, y como ayuda para identificar la variedad de plataformas de algunos de los principales fabricantes, la siguiente tabla identifica los diferentes sistemas por los que han apostado los fabricantes, sistemas actuales y futuras pretensiones [31].

| Fabricante | Plataforma actual | Plataforma anterior | Futura plataforma |
|---|---|---|---|
| Amazon | Fire TV | - | - |
| Apple | Apple TV | - | - |
| Google | Chromecast<br>Android TV | Google TV | - |
| LG | WebOS | NetCast<br>Google TV | - |
| Panasonic | Life+Screen | VIERA Connect<br>VIERA Cast | Firefox OS |
| Phillips | Net TV | - | Android TV |
| Samsung | Samsung Smart TV | Google TV | Tizen OS |
| Sony | Sony Apps | - | Android TV |
| Roku | Roku | Google TV | - |

*Tabla 1. Plataformas principales de Smart TV.*

Por otra parte, en la ilustración 7, se puede apreciar una tabla comparativa de los principales set-top-boxes actuales, comentados anteriormente. En ella se pueden apreciar diferentes aspectos, como especificaciones técnicas, precio y acceso a principales funcionalidades o servicios.

| amazon fireTV |  Amazon Fire TV |  Roku 3 |  Apple TV |  Google Chromecast |
|---|---|---|---|---|
| **List Price** | **$99** | **$99** | **$99** | **$35** |
| **Features** | | | | |
| Voice Search | ✓ | | | |
| HDMI video out (up to 1080p) | ✓ | ✓ | ✓ | ✓ |
| Certified Dolby Digital Plus surround sound | ✓ | | | |
| Optical audio out | ✓ | | ✓ | |
| Processor | Quad-core | Dual-core | Single-core | Single-core |
| Memory | 2 GB | 512 MB | 512 MB | 512 MB |
| Ethernet (wired connectivity) | ✓ | ✓ | ✓ | |
| Wi-Fi | Dual-band/Dual-antenna (MIMO) | Dual-band/Dual-antenna (MIMO) | Dual-band/Dual-antenna | Single-band |
| Remote with no line of sight required | ✓ | ✓ | | |
| **Popular Services** | | | | |
| Netflix | ✓ | ✓ | ✓ | ✓ |
| Amazon Instant Video | ✓ | ✓ | | |
| Hulu Plus | ✓ | ✓ | ✓ | ✓ |
| Crackle | ✓ | ✓ | ✓ | ✓ |
| YouTube | ✓ | ✓ | ✓ | ✓ |
| HBO GO | ✓ | ✓ | ✓ | ✓ |
| Showtime Anytime | ✓ | ✓ | ✓ | ✓ |
| WatchESPN | ✓ | ✓ | ✓ | ✓ |
| Bloomberg TV | ✓ | ✓ | ✓ | |
| Vevo | ✓ | ✓ | ✓ | ✓ |
| Pandora | ✓ | ✓ | | ✓ |
| MLB.TV | ✓ | ✓ | ✓ | ✓ |
| Twitch | ✓ | | | ✓ |
| **Games** | | | | |
| Number of Games | Over 400 | Less than 100 | None | Less than 75 |
| Optional dedicated game controller (sold separately) | ✓ | | | |

*Ilustración 7. Tabla comparativa de los principales set-top-boxes actuales (Enero 2015). Fuente: amazon.com*

## 2.2 Introducción a Android

En esta sección se van a tratar los principales aspectos del sistema operativo Android, como la arquitectura del propio sistema, los componentes más importantes de una aplicación, archivo de manifiesto, los diferentes niveles de API y principales elementos para el desarrollo de interfaces gráficas de usuario. No se pretende realizar un exhaustivo estudio del mismo, sólo resumir los conceptos más importantes que son necesarios para entender la plataforma Android TV.

### 2.2.1 Arquitectura

La arquitectura de Android está basada en una pila que comprende: aplicaciones, sistema operativo, entorno en tiempo de ejecución, middleware, servicios y librerías. Dicha arquitectura podría representarse, aproximadamente, como muestra la siguiente figura [32].

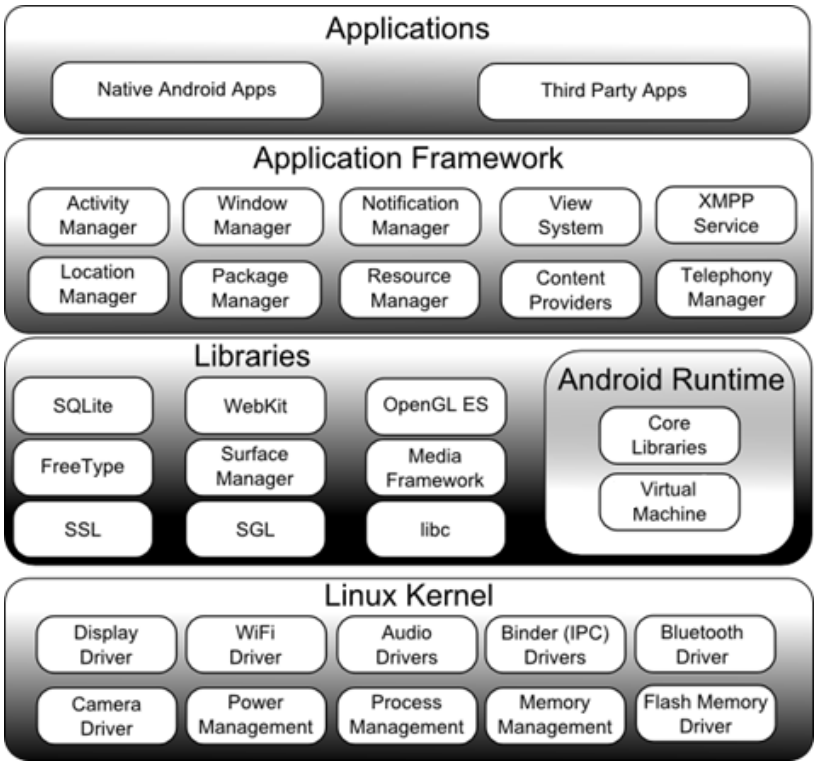


*Ilustración 8. Arquitectura Android. Fuente: techotopia.com*

### Kernel de Linux

El kernel de Linux actúa como middleware, es decir, es una capa de abstracción entre los diferentes componentes hardware de los dispositivos y las capas superiores de la pila. Proporciona la multitarea, servicios del sistema de bajo nivel, como la gestión de memoria, servicios y energía. Además, proporciona la pila de red y los diversos controladores para el hardware, como la pantalla, el Wi-Fi o el audio.

### Entorno en tiempo de ejecución

*Máquina virtual ART:* como se ha comentado anteriormente, el kernel de Linux permite la multitarea, lo que significa que diversos procesos pueden ejecutarse concurrentemente. Sería lógico pensar que cada aplicación Android se ejecuta como un proceso directamente en el kernel de Linux, pero no es así. De hecho, cada aplicación que se ejecuta en un dispositivo Android lo hace dentro de su propia instancia de la máquina virtual ART.

*Librerías Android:* esta categoría abarca aquellas bibliotecas basadas en Java que son específicas para el desarrollo de Android. Algunos ejemplos serían las librerías que facilitan el desarrollo de interfaces de usuario, creación de gráficos o acceso a bases de datos.

### Framework de aplicación

Es un conjunto de servicios que forman colectivamente el entorno en el que se ejecutan y gestionan las aplicaciones de Android. Este marco implementa el concepto

de que las aplicaciones de Android se construyen a partir de componentes reutilizables, intercambiables y reemplazables.

Algunos de los servicios que incluye el framework son:

- *Activity* Manager (gestor de *activities*): controla todos los aspectos del ciclo de vida de la aplicación y la pila de *activities*.
- Proveedores de contenido: permiten a las aplicaciones compartir y publicar datos con otras aplicaciones.
- Gestor de notificaciones: permite que las aplicaciones puedan mostrar notificaciones al usuario.
- Gestor de localización: proporciona el acceso al servicio de localización, permitiendo que la aplicación reciba actualizaciones sobre cambios en la localización.

#### Aplicaciones

Se encuentran en la parte superior de la pila de la arquitectura. Se compone tanto de aplicaciones nativas proporcionadas por el sistema operativo (como el explorador de internet o aplicaciones de correo), como de aplicaciones de terceros instaladas por el usuario.

### 2.2.2 Componentes de una aplicación

En esta parte se van a describir los componentes principales de una aplicación Android y su utilidad.

#### Activity

Una *activity* es un componente de la aplicación que proporciona una pantalla con la que el usuario puede interactuar para realizar alguna acción, como marcar un número de teléfono, sacar una foto o mandar un correo electrónico. En general, casi todas las *activities* interaccionan con el usuario por lo que la clase *activity* permite insertar una interfaz de usuario (UI). Suelen presentarse a los usuarios en pantalla completa, pero también se pueden utilizar como ventanas flotantes o embebidas dentro de otra *activity* [33] [34].

Android guarda un historial con las diferentes *activities* que ha visitado el usuario. Dicho historial se almacena en la pila de *activities* (*Activity* Stack o Back Stack), que permite volver a *activities* anteriores.

#### Servicios

Un servicio es un componente de la aplicación que puede llevar a cabo grandes operaciones en segundo plano, sin necesidad de proporcionar una interfaz al usuario.

Otro componente de la aplicación puede iniciar un servicio, que se mantiene en segundo plano, incluso si el usuario cambia de aplicación. Además, un componente se puede unir a un servicio para interactuar con él, e incluso llevar a cabo una comunicación entre procesos (IPC). Por ejemplo, un servicio puede manejar transacciones de red, reproducir música o interactuar con un proveedor de contenido, todo ello en segundo plano [35].

Un servicio, esencialmente, puede tener dos estados:

- Iniciado (started): cuando un componente de la aplicación ejecuta dicho servicio. Una vez iniciado, se puede mantener en segundo plano indefinidamente, incluso si el componente que lo inició es destruido.
- Unido o ligado (bound): cuando un servicio ha sido unido o ligado a un componente de la aplicación. En este caso, el servicio ofrece una interfaz cliente-servidor que permite al componente interactuar con dicho servicio, mandar peticiones y recibir respuestas.

#### Receptores *broadcast* (*broadcast receivers*)

Un receptor *broadcast* se encarga de recibir y reaccionar ante la detección de un mensaje *broadcast* de otra aplicación o del propio sistema. Por ejemplo, una aplicación puede mandar un mensaje *broadcast* para que otras aplicaciones sepan que ha ocurrido cierta acción (se ha completado una descarga o la batería está baja) [36].

#### Proveedores de contenido (*Content providers*)

Los proveedores de contenido gestionan el acceso a un conjunto estructurado de datos. Encapsulan dichos datos y proporcionan mecanismos de seguridad. Se encarga de suministrar datos de una aplicación a otra bajo demanda.

Generalmente, cada aplicación Android ejecuta sus propios procesos con sus propios permisos, lo que mantiene ocultos los datos de dicha aplicación a otras. Pero a veces se necesita compartir datos entre diferentes aplicaciones y es aquí donde los proveedores de contenidos son muy útiles.

Su comportamiento es muy similar a una base de datos, donde puedes realizar consultas, editar, crear o eliminar contenido mediante métodos como: *insert, update o delete* [37].

#### *Intent*

Un *intent* es un objeto que se utiliza como “mensajero” para comunicar diferentes acciones entre los diferentes componentes de una aplicación. Existen dos tipos, explícitos, que especifican el componente que se va a ejecutar, o implícitos, en los que no se especifica, pero se declara una acción concreta a realizar. Algunos de los más

habituales son los *intents* para lanzar una *activity* o servicio nuevo, o para mandar un mensaje *broadcast* [38].

### El archivo de manifiesto (AndroidManifest.xml)

Todas las aplicaciones deben tener un archivo de manifiesto en el directorio raíz del proyecto. Este archivo contiene información esencial, que el sistema debe leer antes de ejecutar propiamente la aplicación. En él se definen diferentes funcionalidades y requerimientos, entre los que se pueden destacar [39]:

- Los componentes de la aplicación: *activities*, servicios, receptores *broacast* y proveedores de contenido.
- El nombre del paquete Java que contiene la aplicación.
- Los permisos que debe tener la aplicación para poder acceder a partes protegidas de la API y poder interactuar con otras aplicaciones.
- Nivel mínimo del API de Android que requiere la aplicación.

### Elementos principales de interfaces gráficas

En este apartado se van a describir los principales elementos para la creación de interfaces de usuario de Android. Es una parte fundamental de una aplicación ya que compone la parte visual con la que el usuario va a interactuar.

#### View

La clase *View* representa el componente más básico a la hora de crear interfaces gráficas de usuario. Ocupa una parte rectangular de la pantalla y actúa como contenedor para diferentes elementos gráficos como pueden ser botones, campos de texto, imágenes, etc. Además, se encargan de manejar los diferentes eventos de los elementos que contenga. Ejemplos destacados de clases View: *button*, *TextView*, *ListView*, *ImageView*, etc.

Cabe destacar, también, la clase *ViewGroup,* una View específica que permite contener varias *Views*. Es la base para la creación de *layouts*, comentados a continuación [40].

#### *Layouts*

Un *layout* define la estructura visual de la interfaz de usuario. Contiene los elementos gráficos y controla su comportamiento y posición, incluso puede contener otro *layout*. Desciende de la clase *View*.

La forma más habitual de definir un *layout* es a través de un archivo XML, en el que se declara el tipo de *layout* y los diferentes elementos gráficos que va a contener [41].

Los *layouts* más comunes son los siguientes:

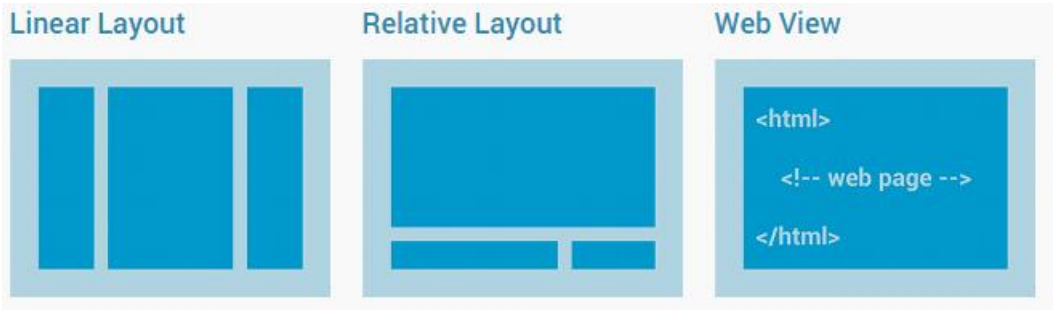


*Ilustración 9. Ejemplos de layouts más comunes. Fuente: developer.android.com*

Por otra parte, se pueden construir *layouts* dinámicos gracias a un elemento Android llamado *Adapter*, que obtiene datos de la fuente que se le indique y los convierte en diferentes elementos tipo View, que conforman el *layout* final. Los más comunes son:

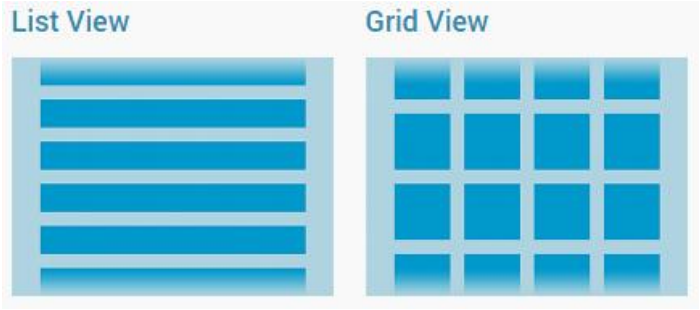


*Ilustración 10. Ejemplo layouts dinámicos para la creación de listas con ListView y tablas con GridView. Fuente: developer.android.com*

### *Fragment*

Es uno de los componentes principales de las interfaces de usuario para Android TV. Representa una porción de la interfaz de usuario de una *activity*. Se pueden combinar varios *fragments* en una *activity* para crear una interfaz multi-panel y reutilizar un *fragment* en otras *activities*.

El *fragment* debe ir siempre embebido dentro de una Activity y su ciclo de vida va ligado al de la propia a*ctivity*. Si la *activity* se pausa o se destruye, el *fragment* experimenta lo mismo [42].

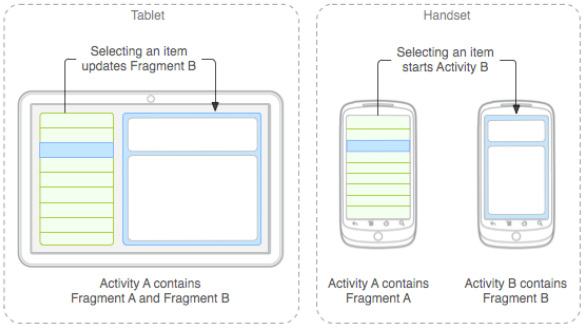


*Ilustración 11. Ejemplo de interfaces basadas en fragments combinados en la misma activity en una tablet o separados en dos activities en smartphone. Fuente: developer.android.com*

### 2.2.3 Niveles de API de Android

El nivel de API es un valor entero que identifica de forma unívoca el framework ofrecido por la versión de una plataforma Android. Dicho framework facilita la interacción con el sistema Android y se compone, principalmente, de: un conjunto de paquetes y clases, elementos y atributos XML para la declaración del archivo de manifiesto y los diferentes permisos que las aplicaciones pueden requerir.

La mayoría de los cambios de una nueva versión del API son aditivos, para permitir la retro compatibilidad con versiones anteriores e incluyen nuevas funcionalidades o modifican alguna ya existente.

A continuación, en la tabla 2, se muestran las diferentes versiones de Android hasta la fecha, junto con el nivel de API correspondiente y su nombre comercial [43].

En este caso, Android TV trabaja sobre la versión 21 del API, la más reciente hasta la fecha, Android 5.0 Lollipop.

| Versión de la plataforma | Nivel API | Nombre |
|---|---|---|
| Android 5.0 | 21 | Lollipop |
| Android 4.4W | 20 | Kitkat_Watch |
| Android 4.4 | 19 | Kitkat |
| Android 4.3 | 18 | Jelly Bean MR2 |
| Android 4.2, 4.2.2 | 17 | Jelly Bean MR1 |
| Android 4.1, 4.1.1 | 16 | Jelly Bean |
| Android 4.0.3, 4.0.4 | 15 | Ice Cream Sandwich MR1 |
| Android 4.0, 4.0.1, 4.0.2 | 14 | Ice Cream Sandwich |
| Android 3.2 | 13 | Honeycomb MR2 |
| Android 3.1.X | 12 | Honeycomb MR1 |
| Android 3.0.X | 11 | Honeycomb |
| Android 2.3.4, 2.3.3 | 10 | Gingerbread MR1 |

| | | |
|---|---|---|
| Android 2.3.2, 2.3.1, 2.3 | 9 | Gingerbread |
| Android 2.2.X | 8 | Froyo |
| Android 2.1.X | 7 | Eclair MR1 |
| Android 2.0.1 | 6 | Eclair 0 1 |
| Android 2.0 | 5 | Eclair |
| Android 1.6 | 4 | Donut |
| Android 1.5 | 3 | Cupcake |
| Android 1.1 | 2 | BASE 1.1 |
| Android 1.0 | 1 | BASE |

*Tabla 2. Niveles de API Android hasta el momento. Fuente: developers.google.com*

# Capítulo 3: Configuración y primeros pasos.

En este capítulo se van a explicar los requisitos necesarios para comenzar a desarrollar en Android TV.

Por otra parte, incluye una guía que recoge una serie de consideraciones que se deben tener en cuenta a la hora de desarrollar para un dispositivo como es una televisión y los principales elementos de una aplicación de estas características.

## *3.1 Requisitos*

Los dos componentes principales para crear una aplicación que funcione en una TV son los siguientes:

- *Activity* para la TV (requerido): en el archivo manifest del proyecto hay que crear una *activity* cuyo objetivo es ejecutarse en una TV.

- Librerías para el soporte de TV (opcionales): existen diversas librerías que facilitan la creación de interfaces pensadas para una TV.

### 3.1.1 Prerrequisitos

- Versión **24.0.0** o superior del SDK Tools.
- Versión de **Android 5.0 (API 21)** o superior.
- Crear o modificar un proyecto existente cuyo **"target"** sea **Android 5.0 (API 21)** o superior.

**Declaración de una *Activity* para TV**

Una aplicación pensada para funcionar en un dispositivo de TV tiene que tener declarada en el manifest una *activity* para TV utilizando el "intent-filter" **CATEGORY_LEANBACK_LAUNCHER.** Dicho filtro identifica que la aplicación está pensada para una TV, permitiendo así que sea considerada como una app de TV en Google Play. Dicho filtro también indica qué *activity* debe lanzarse cuando la app se ejecute en una TV.

**Nota:** si no se incluye el "intent-filter" de **CATEGORY_LEANBACK_LAUNCHER**, la aplicación no será visible en el Google Play si se ejecuta desde una TV. Además, si se carga la app mediante herramientas de desarrollo, no aparecerá en la interfaz de la TV.

```
<application>
  ...
  <activity
    android:name="com.example.android.MainActivity"
    android:label="@string/app_name" >
```

```
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>

<activity
    android:name="com.example.android.TvActivity"
    android:label="@string/app_name"
    android:theme="@style/Theme.Leanback">

    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LEANBACK_LAUNCHER" />
    </intent-filter>

  </activity>
</application>
```

En el ejemplo anterior se puede apreciar que la segunda a*ctivity* es la destinada a ejecutarse en una TV, ya que contiene el filtro **LEANBACK_LAUNCHER** comentado anteriormente.

**Librerías de soporte para TV**

El SDK de Android incluye librerías de soporte pensadas para su uso en aplicaciones de TV. Estas librerías proporcionan APIs y herramientas de interfaz de usuario para ser usadas en dispositivos de TV. Existen tres grupos de librerías:

- Librería **Leanback v17**: proporciona herramientas para facilitar el desarrollo de interfaces en TV, principalmente pensada para apps que reproducen contenido multimedia.
- Librería **Recyclerview v7**: proporciona clases para facilitar la visualización de grandes listas de una manera más eficiente. Muchas de las clases de la librería anterior dependen de clases de esta librería.
- Librería **Cardview v7**: contiene herramientas para facilitar la presentación de diferente información en forma de "cartas" o paneles, como, por ejemplo, elementos multimedia con imágenes y descripciones.

**Nota:** Estas librerías no son obligatorias, pero su uso es muy recomendado, particularmente para aplicaciones como catálogos multimedia.

## *3.2 Consideraciones a la hora de diseñar para TV.*

Para diseñar aplicaciones pensadas para utilizarlas en una televisión, hay que tener en cuenta una serie de aspectos, ya que la visualización y el uso de dichas aplicaciones no es igual que en otros dispositivos como móviles o tablets.

**Optimización de layouts**

Hay que tener en cuenta que el usuario va a estar sentado a una distancia del televisor de unos 3 metros, por ello hay que tener este dato en mente a la hora de diseñar la interfaz gráfica de la aplicación.

**Disposición horizontal/apaisada**

Los televisores siempre están en disposición horizontal o apaisada, a diferencia de los móviles o tablets. Para construir diseños optimizados para TV es recomendable seguir los siguientes pasos:

- Los controles para la navegación deben estar a la izquierda o derecha de la pantalla, dejando así, el mayor espacio vertical posible para el contenido.
- Se debe dividir ese espacio en diferentes secciones, utilizando para ello el elemento Android denominado "Fragment" y utilizar diseños de tipo rejilla en lugar de listas (GridView en lugar de ListView).
- Utilizar diseños y medidas relativos, no absolutos, para que los elementos ajusten su tamaño, relación de aspecto y densidad de píxeles, acordes a un televisor.
- Añadir suficiente margen a los diversos componentes para evitar solapamiento.


*Ilustración 12. Ejemplo con controles a la izquierda y contenido distribuido en un diseño tipo rejilla. Fuente: developer.android.com*

**Textos y controles fácilmente visibles**

Los textos y los controles deben ser muy fáciles de ver en una TV, ya que se va a utilizar a cierta distancia. Es aconsejable tener en cuenta los siguientes puntos:

- Dividir los textos en fragmentos más pequeños para que sean más fáciles de divisar.

- Un estilo sencillo para leer mejor los textos es utilizar un color claro sobre un fondo oscuro.
- No se deben utilizar tipografías muy finas o estrechas. Se aconseja el uso de fuentes *sans-serif* y el uso de *anti-aliasing*.
- Utilizar los tamaños estándar para tipografías en Android. En función de la importancia del texto existen unos tamaños estándar, 14pt para párrafos y texto en botones, 20pt para títulos, 16pt para subtítulos.
- Hay que asegurarse que todos los componentes y elementos son visibles. Para ello, se aconseja el uso de diseños relativos, mejor que absolutos, y el uso de unidades independientes de la densidad de pixel en lugar de unidades de pixel absolutas.

**Diseño para grandes pantallas con gran densidad de píxeles**

Actualmente, las resoluciones más comunes son 720p, 1080i y 1080p. Lo más recomendable es diseñar las aplicaciones para 1080p y, en caso, de no soportar dicha resolución, Android realiza una disminución en la escala (eliminando píxeles). En general, este proceso no degrada la calidad de la imagen.

**Optimización de la navegación**

Un aspecto muy importante de la experiencia de usuario a la hora de manejar un televisor es el mando a distancia. Hay que tener en cuenta que, a diferencia de otros dispositivos, la forma de interactuar con una TV es a través de un control remoto.

**Navegación mediante D-Pad**

En una TV, el usuario va a utilizar un D-Pad o el típico control por flechas (arriba, abajo, izquierda y derecha), lo que limita en gran medida la libertad de movimiento. Por ello, a la hora de diseñar una aplicación se deben seguir estos puntos:

- Hay que asegurarse de que es posible navegar fácilmente por todos los controles que aparecen en la pantalla.
- En el caso de que aparezcan listas, se debe poder hacer scroll hacia arriba y abajo mientras se seleccionan los diferentes elementos.
- El movimiento entre los diferentes controles debe ser predecible.

En general, Android maneja la navegación entre los diferentes controles sin problema, pero si no se acomoda a la necesidad o gusto del programador, se puede modificar y configurar unos controles específicos.

**Proporcionar indicaciones visuales claras en selecciones y elementos importantes**

Se deben utilizar colores apropiados para resaltar los elementos seleccionados en cada momento. Esto facilita mucho la navegación al usuario. Se debe asegurar cierto margen entre los elementos para que aquel que esté seleccionado sea claramente visible.

**Diseño para proporcionar una navegación sencilla**

Los usuarios deben ser capaces de navegar a cualquier elemento de la interfaz con un par de clicks en el control remoto. La navegación debe ser lo más sencilla e intuitiva posible. Para ciertas acciones que no sean lo suficiente intuitivas, se debe proporcionar al usuario una ayuda en pantalla.

Se debe intentar predecir la siguiente acción que va a llevar a cabo el usuario y hacer que sea lo más sencilla posible. Por ello la interfaz de navegación no se debe quedar escasa, si se necesita más espacio se pueden usar más fragmentos para subdividir dicha sección de navegación.

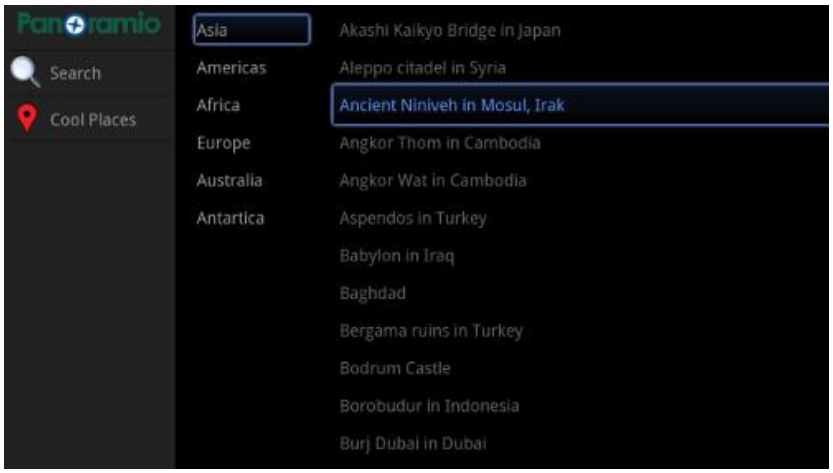


*Ilustración 13. Ejemplo de subdivisiones del panel de navegación. Fuente: developer.android.com*

## *3.3 Construyendo aplicaciones para TV.*

En esta sección se van a explicar los detalles más relevantes de los diferentes elementos más característicos de las aplicaciones orientadas a TV.

### 3.3.1 Aplicaciones para la reproducción de contenidos

En una TV, uno de los tipos más característicos de aplicaciones son aquellas destinadas a la búsqueda y reproducción de contenidos multimedia. Es importante proporcionar a los usuarios un uso lo más rápido, eficiente e intuitivo posible. Tal y como se comentó anteriormente, Android proporciona una serie de clases que facilitan dicha tarea.

**Creación navegador tipo catálogo**

Las aplicaciones multimedia suelen contener un gran número de elementos, por ello, la forma más sencilla de presentarlos suele ser mediante el uso de catálogos y la posibilidad de navegar en ellos. Elementos importantes de un catálogo:

- Layout: el layout o diseño más recomendable es **"BrowseFragment"** de la librería **leanback**, que permite la creación de un layout principal para la

navegación por categorías y filas de elementos multimedia con un mínimo de código.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  >

  <fragment
      android:name="android.support.v17.leanback.app.BrowseFragment"
      android:id="@+id/browse_fragment"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      />
</LinearLayout>
```

- Visualización de listas de elementos multimedia: el propio **"BrowseFragment"** permite definir y visualizar categorías y contenido multimedia de un catálogo mediante el uso de **Adaptadores** (adapters) y **Presentadores** (presenters). Los adapters proporcionan la conexión a una fuente local o en línea donde se encuentre la información del catálogo. Por su parte, los presenters, obtienen la información de los elementos multimedia y facilitan su visualización.

**Actualización del fondo**

Para que la app sea más atractiva, una buena técnica es ir actualizando el fondo a medida que el usuario navega por dicha app.

Para ello, la librería **Leanback** contiene una clase llamada **BackgroundManager** que permite modificar el fondo del *activity* de la app. Una práctica muy común consiste en actualizar el fondo automáticamente en base a la navegación y selección del usuario. Para ello se debe utilizar un listener, más concretamente, **OnItemViewSelectedListener,** para capturar los eventos de selección por parte del usuario y cambiar el fondo en consecuencia.

### 3.3.2 Construyendo la vista de detalles

La librería v17 leanback proporciona clases para visualizar información adicional sobre elementos multimedia, como una descripción u opiniones, o la posibilidad de realizar diversas acciones como adquirir ese elemento o reproducirlo.

**Construir un "presenter" para mostrar detalles**

Como se ha comentado anteriormente, el objeto utilizado para visualizar elementos en la interfaz son los presentadores (presenters). Entre ellos, podemos encontrar la clase **AbstractDetailsDescriptionPresenter** para el propósito actual. Todo lo que hay que hacer es implementar el método **onBindDescription():**

```
public class DetailsDescriptionPresenter
        extends AbstractDetailsDescriptionPresenter {


protected void onBindDescription(ViewHolder viewHolder, Object  itemData)
{
MyMediaItemDetails details = (MyMediaItemDetails) itemData;

viewHolder.getTitle().setText("Título");
viewHolder.getSubtitle().setText("Subtítulo");
viewHolder.getBody().setText("Sinopsis");}

}
```

**Extender el *fragment* Details**

Es recomendable extender la clase **DetailsFragment** para visualizar contenido multimedia, ya que proporciona contenido adicional, como previsualización de imágenes y acciones a llevar a cabo sobre dichos elementos multimedia.

Los *fragments* como DetailsFragment deben estar dentro de una *activity* para poder utilizarlos para visualizar contenidos. De esta manera se puede invocar la vista de detalles mediante un **Intent.**

### 3.3.3 Visualizar contenido "Playing Now"

Normalmente, las apps de TV permiten la reproducción de música u otros contenidos de fondo, pero se debe proporcionar un modo de regresar a dicha reproducción para pausar o cambiar canción. Android soporta dicha funcionalidad en las apps de TV mediante las "cartas" o paneles ***"Now Playing"*** en la pantalla de inicio.

Este panel o carta se muestra en la pantalla de inicio, en la misma fila que las recomendaciones de usuario e incluye información sobre el archivo multimedia, como el álbum, título e icono de la app. Si el usuario lo selecciona, el sistema abre la app que contiene dicha sesión multimedia.

**Comenzar una sesión multimedia**

Una app de reproducción multimedia puede ejecutarse como una ***activity*** o como un **servicio.** Para el caso de la reproducción en segundo plano, se debe ejecutar como servicio, ya que seguirá ejecutándose incluso después de destruir la *activity* que lo ejecutó. El servicio utilizado en este caso es **MediaBrowserService.**

```
mSession = new MediaSession(this, "MusicService");
mSession.setCallback(new MediaSessionCallback());
mSession.setFlags(MediaSession.FLAG_HANDLES_MEDIA_BUTTONS |
        MediaSession.FLAG_HANDLES_TRANSPORT_CONTROLS);
```

```
setSessionToken(mSession.getSessionToken());
```

El panel Now Playing solo mostrará contenido multimedia de una sesión que tenga configurado el flag **FLAG_HANDLES_TRANSPORT_CONTROLS.**

**Visualizar panel "Now Playing"**

Este panel se muestra tras llamar al método **setActive(true)** si esa sesión es la de mayor prioridad en el sistema. Además, la app debe solicitar el **"audio focus":**

```
private void handlePlayRequest() {

    tryToGetAudioFocus();

    if (!mSession.isActive()) {
        mSession.setActive(true);
    }
...
```

El panel se elimina si ejecuta el método **setActive(false)** o si otra app inicia la reproducción de contenido multimedia.

**Actualizar el estado de la reproducción**

Como en cualquier app de reproducción multimedia, se debe actualizar el estado del reproductor, de modo que muestre la información del contenido actual:

```
private void updatePlaybackState() {
    long position = PlaybackState.PLAYBACK_POSITION_UNKNOWN;
    if (mMediaPlayer != null && mMediaPlayer.isPlaying()) {
        position = mMediaPlayer.getCurrentPosition();
    }
    PlaybackState.Builder stateBuilder = new PlaybackState.Builder()
            .setActions(getAvailableActions());
    stateBuilder.setState(mState, position, 1.0f);
    mSession.setPlaybackState(stateBuilder.build());
}
private long getAvailableActions() {
    long actions = PlaybackState.ACTION_PLAY |
            PlaybackState.ACTION_PLAY_FROM_MEDIA_ID |
            PlaybackState.ACTION_PLAY_FROM_SEARCH;
    if (mPlayingQueue == null || mPlayingQueue.isEmpty()) {
        return actions;
    }
    if (mState == PlaybackState.STATE_PLAYING) {
        actions |= PlaybackState.ACTION_PAUSE;
    }
    if (mCurrentIndexOnQueue > 0) {
        actions |= PlaybackState.ACTION_SKIP_TO_PREVIOUS;
    }
    if (mCurrentIndexOnQueue < mPlayingQueue.size() - 1) {
        actions |= PlaybackState.ACTION_SKIP_TO_NEXT;
```

```
        }
        return actions;
    }
```

### 3.3.4 Facilitar la búsqueda de contenido a usuarios

Cuando se interactúa con una TV, normalmente, los usuarios prefieren acceder a contenidos lo más rápido posible, en los menos pasos posibles. Para ello, una muy buena práctica es ofrecer contenidos recomendados para el usuario al iniciar la TV. Agregando contenido de nuestro catálogo, puede ayudar a que el usuario acceda a la app.

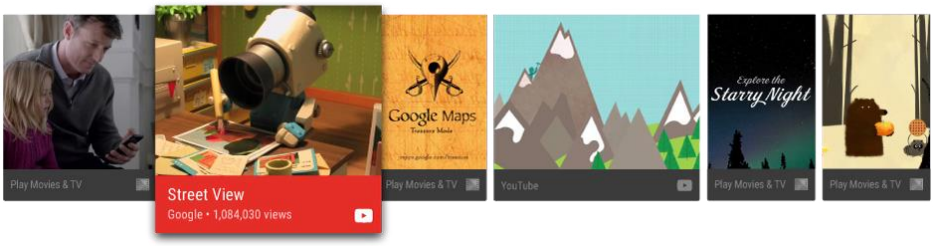


*Ilustración 14. Ejemplo de contenidos recomendados. Fuente: developer.android.com*

**Creando un servicio de recomendaciones**

Los contenidos que conforman la sección de recomendaciones se generan en un proceso en segundo plano. Para actualizar dicha sección con contenidos del catálogo de nuestra app, se debe crear un servicio que periódicamente agregue nuevos contenidos.

Ejemplo de **IntentService** para crear un servicio de recomendaciones:

```
public class RecommendationsService extends IntentService {
    private static final int MAX_RECOMMENDATIONS = 3;

    public RecommendationsService() {
        super("RecommendationService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        MovieDatabase database =
MovieDatabase.instance(getApplicationContext());
        List recommendations = database.recommendations();

        int count = 0;

        try {
            for (Movie movie : recommendations) {
                buildRecommendation(getApplicationContext(), movie);
```

```
                if (++count >= MAX_RECOMMENDATIONS) {
                    break;
                }
            }
        } catch (IOException e) {
            Log.e(TAG, "Error", e);
        }
    }
}
```

Para que funcione, hay que registrar dicho servicio en el **manifest** de la app.

```
<manifest ... >
  <application ... >
    <service android:name=".RecommendationsService"
             android:enabled="true" android:exported="true"/>
  </application>
</manifest>
```

Una vez creado el servicio de recomendaciones, debe empezar a generar contenidos recomendados y pasarlos al sistema, que los recibe como objetos **Notification.**

**Ejecución del servicio de recomendaciones**

Este servicio debe ejecutarse cada cierto tiempo para ir actualizando la sección de recomendaciones. Para ello se debe crear una clase que, mediante un temporizador, invoque dicho servicio.

*Nota: ejemplo en la web con un **BroadcastReceiver,** que ejecute periódicamente el servicio cada 12 horas.*

El **BroadcastReceiver,** debe ejecutarse justo después de la puesta en marcha de la TV. Para ello, hay que agregarlo en el manifest con un filtro para que se ejecute después del proceso de arranque:

```
<manifest ... >
  <application ... >
    <receiver android:name=".BootupReceiver" android:enabled="true"
              android:exported="false">
      <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
      </intent-filter>
    </receiver>
  </application>
</manifest>
```

### 3.3.5 Creación de Apps con contenido en directo

Los programas y canales en directo son uno de los puntos fuertes de las TVs. Android soporta la recepción y reproducción de contenido en vivo gracias al TV input framework en Android 5.0 (API 21). Este framework proporciona un método unificado para la recepción de audio y vídeo procedentes de fuentes hardware (como entradas HDMI) y fuentes software (como vídeos online).

El framework permite a los desarrolladores definir las fuentes de entrada mediante la implementación de un servicio de entrada. Este servicio proporciona una lista de canales y programas al proveedor de TV (TV Provider). La app de la TV recibe dicha lista y la muestra al usuario. Cuando el usuario selecciona un canal, la app crea una sesión para el servicio de entrada de TV asociado mediante el "TV Input Manager" y le indica al servicio de la TV que visualice y reproduzca dicho canal.

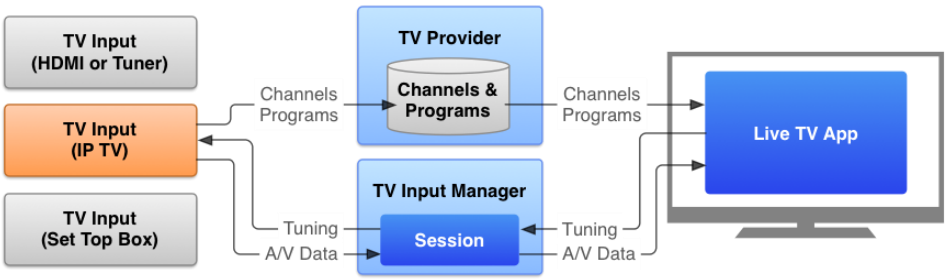


*Ilustración 15. Diagrama funcional del framework de TV. Fuente: developer.android.com*

# Capítulo 4: Diseño y desarrollo de la aplicación.

En este capítulo se detallarán diferentes aspectos sobre el diseño de la aplicación, con los diferentes diseños de cada una de las pantallas que conforman dicha aplicación, una breve descripción de los requisitos que debe cumplir, diagramas de las clases que la componen y una explicación de los apartados más importantes del desarrollo de la aplicación.

## *4.1 Diseño de la aplicación.*

En esta sección se van a detallar los diferentes diseños de cada una de las pantallas que conforman la aplicación completa.

En todas ellas, el diseño es lo más liviano posible, a la par de estar orientado al tamaño de un televisor, y facilitando en todo momento la navegación por parte de un usuario.

### 4.1.1 Pantalla principal

Esta primera pantalla muestra las diferentes categorías, según los géneros de las películas, y todos los elementos de cada una de dichas categorías. En la parte superior izquierda se puede apreciar un icono de búsqueda.

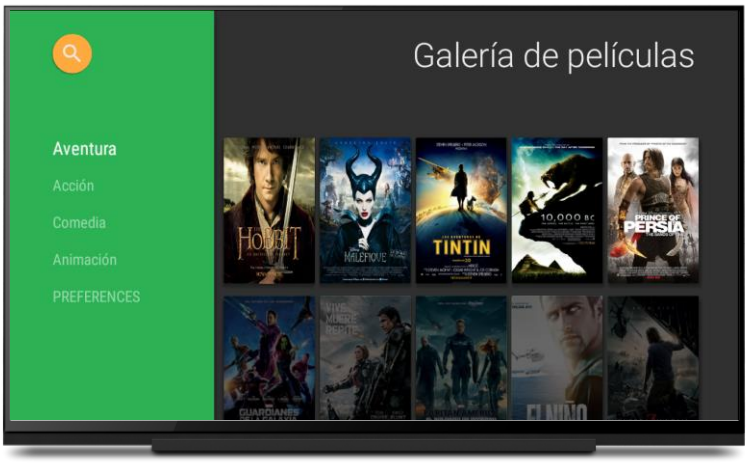


*Ilustración 16. Diseño de pantalla principal.*

### 4.1.2 Pantalla de detalles

Se visualiza contenido detallado de la película que haya seleccionado el usuario. Debe contener una imagen de portada, con el título, breve sinopsis y una opción para acceder a la visualización del tráiler.

Además, en la parte inferior, se muestra un listado de películas relacionadas.


*Ilustración 17. Vista de detalles.*

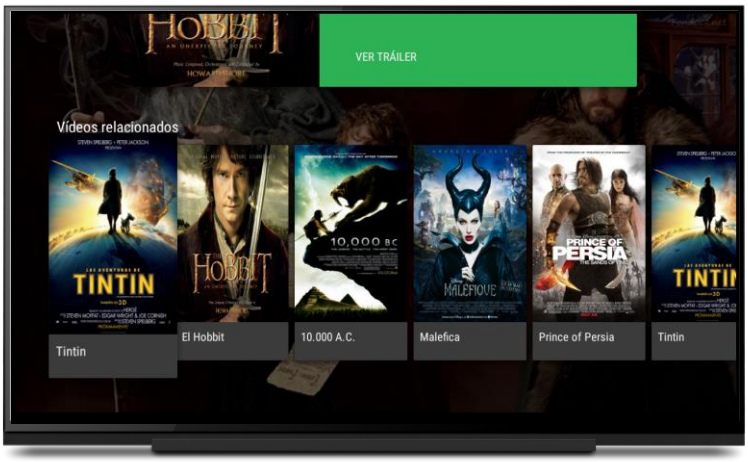


*Ilustración 18. Listado de películas relacionadas con la seleccionada.*

### 4.1.3 Pantalla de reproducción

Esta pantalla solo contiene un reproductor para visualizar el vídeo asociado a la película seleccionada. En este caso, se ha utilizado un diseño proporcionado por Google, que incluye un reproductor estándar.

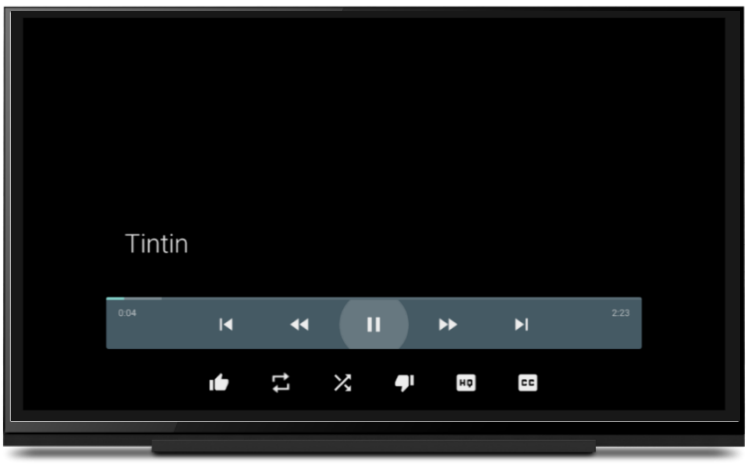


*Ilustración 19. Pantalla con el reproductor de vídeo.*

### 4.1.4 Pantalla de búsqueda

Contiene un cuadro de texto para introducir el texto a buscar y muestra los resultados en caso que existan.

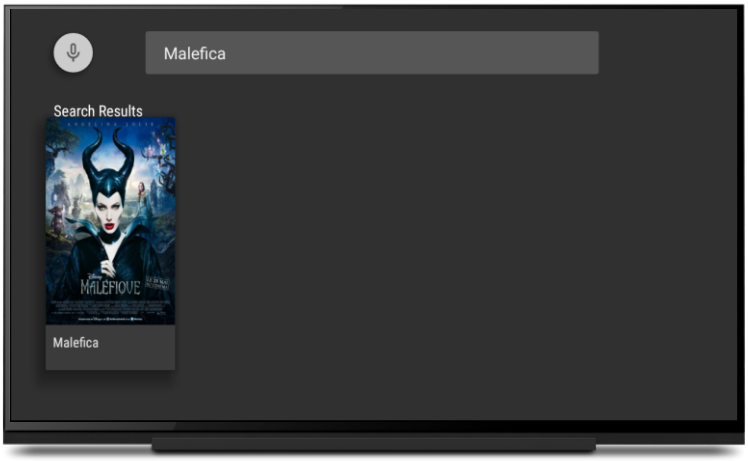


*Ilustración 20. Pantalla con el buscador de contenido.*

### 4.1.5 Pantalla con vista de rejilla

Muestra el contenido completo del catálogo en una vista tipo rejilla.

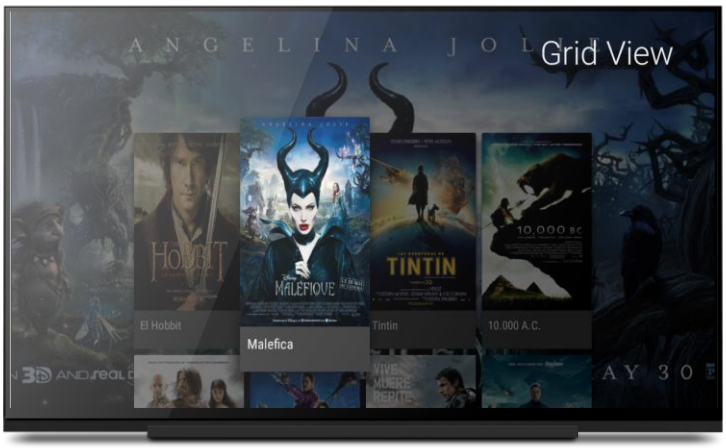


*Ilustración 21. Vista de rejilla.*

## *4.2 Requisitos.*

En esta sección está destinada para la explicación de los diferentes requisitos que debe cumplir la aplicación, acorde a los objetivos que se plantearon inicialmente.

Por la naturaleza del proyecto, ya que no se trata exclusivamente del desarrollo de una aplicación específica, sino que se divide en una primera parte para documentar e investigar una tecnología nueva como es Android TV y una segunda parte orientada a probar y desarrollar una aplicación sencilla en base a dicha investigación, no existen unos requisitos iniciales propiamente hablando.

Sin embargo, a raíz de la previa documentación e investigación, se pueden definir unos puntos o requisitos básicos que una aplicación propia para un televisor debe cumplir, incluyendo una serie de requisitos necesarios para el correcto funcionamiento de la aplicación propuesta.

A continuación de definen esos requisitos.

### 4.2.1 Requisitos funcionales

En esta sección se van a especificar los diferentes requisitos en cuanto a funcionalidad que la aplicación debe cumplir.

- La aplicación debe cargar una serie de categorías, con sus contenidos (películas) correspondientes a cada una de ellas, en una vista tipo catálogo.
- Dentro de cada categoría se debe poder navegar por las diferentes películas.
- Dentro de cada categoría se debe poder navegar a otras categorías sin necesidad de regresar y seleccionar categoría nueva.
- Se debe proporcionar un apartado para realizar búsquedas de contenido dentro de la aplicación.
- Al seleccionar alguna de las películas, debe cargar una nueva pantalla para la vista de detalles.

- La vista de detalles debe contener una portada, título, sinopsis y acceso al tráiler de la película seleccionada.
- En la vista de detalles, se debe proporcionar recomendaciones de contenido relacionado con la película seleccionada.
- Dentro de la vista de detalles, si se pulsa *atrás,* se debe regresar a la pantalla principal con el catálogo.
- Al acceder a la visualización del tráiler, se debe mostrar una nueva pantalla con un reproductor para visualizar el vídeo.
- Dentro de la pantalla de reproducción, si se pulsa *atrás*, se debe regresar a la vista de detalles.
- En la pantalla de búsqueda, se debe proporcionar un apartado para escribir el texto a buscar y mostrar resultados obtenidos, en caso de que existan. Si no se encuentran resultados, se debe avisar con algún tipo de mensaje o alerta, fácilmente visible por el usuario.

### 4.2.2 requisitos de interfaz de usuario

En esta sección se van a detallar los requisitos que debe cumplir la aplicación a nivel de interfaz de usuario, ya que dicha interfaz debe estar orientada para su visualización en un televisor y proporcionar una navegación lo más sencilla e intuitiva posible, teniendo en cuenta que el control remoto, por lo general, cuenta con cuatro flechas de dirección, botón para seleccionar y botón para regresar.

- En la pantalla principal, con el catálogo, se debe navegar por las categorías fácilmente, con las flechas de dirección y resaltar en todo momento qué categoría está seleccionada.
- En el caso de la navegación por las películas, se debe cumplir lo mismo que el requisito anterior: navegación sencilla con los botones de dirección, además de acceso a la película con el botón de seleccionar.
- El acceso al buscador de contenidos debe ser intuitivo y fácil a través delas flechas de dirección.
- En la vista de detalles, se debe acceder fácilmente a la opción que redirige a la visualización del tráiler.
- En el reproductor, los diferentes controles del vídeo deben ser fácilmente seleccionables por parte del usuario y resaltar en todo momento cuál se está seleccionando.

A continuación se muestra un diagrama donde se expone la interacción final del usuario con las diferentes pantallas de la aplicación.

## Interacción final con la aplicación

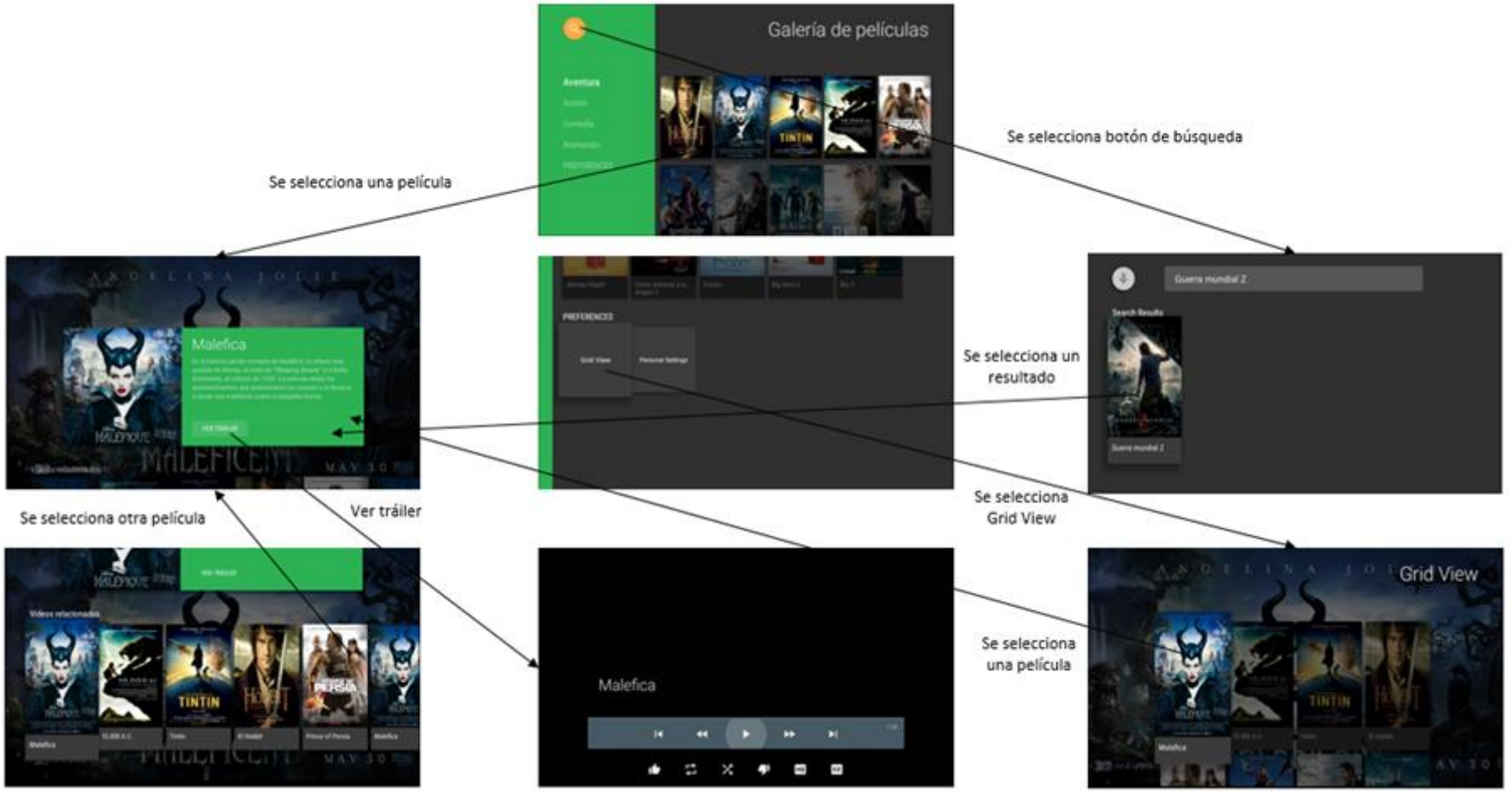


*Ilustración 22. Interacción final con la aplicación.*

## *4.3 Diagrama de clases.*

A continuación se muestran los diferentes diagramas de clases que componen la aplicación final, desarrollados con el software *Enterprise Architect 11*.

**Clase Movie y MovieList para obtención de contenido**

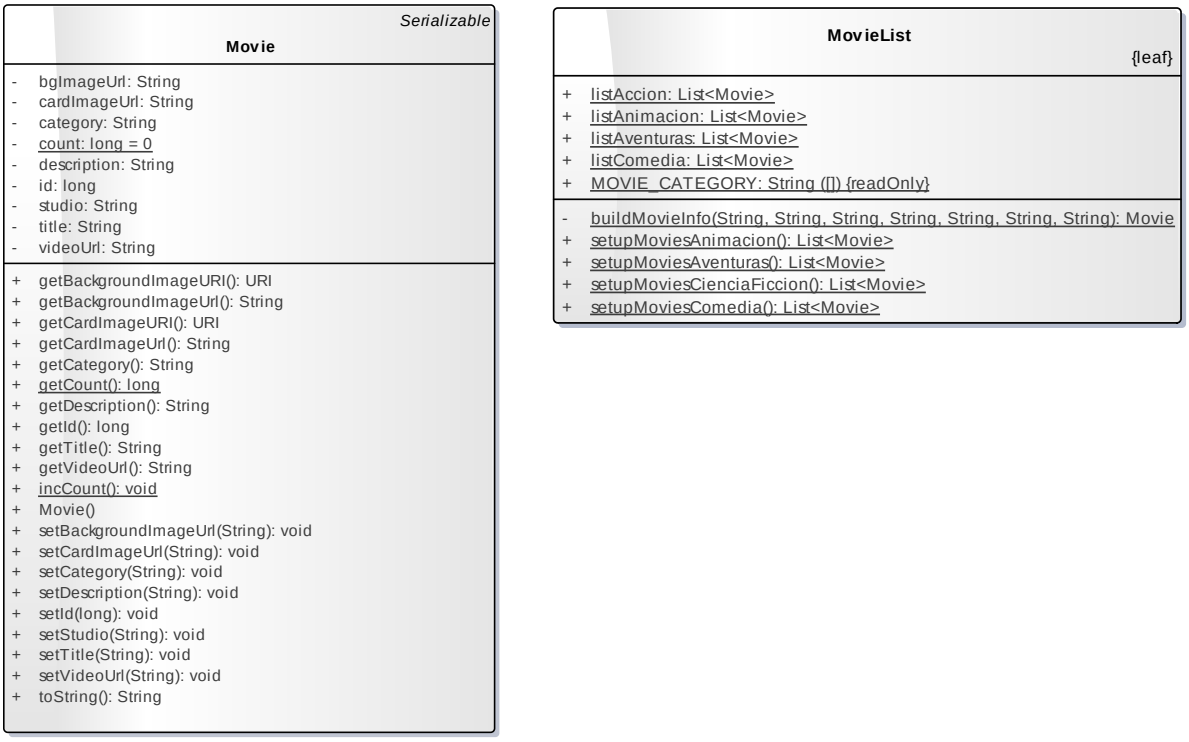


*Ilustración 23. Diagramas clases Movie y MovieList.*

La ilustración 23 muestra el diagrama de la clase *Movie,* que representa cada elemento película del catálogo, con sus atributos y métodos, y el diagrama de la clase *MovieList*, que contiene los métodos para generar los elementos *Movie* y la creación de las diferentes listas de películas por categorías.

### *Fragment* principal – vista de catálogo

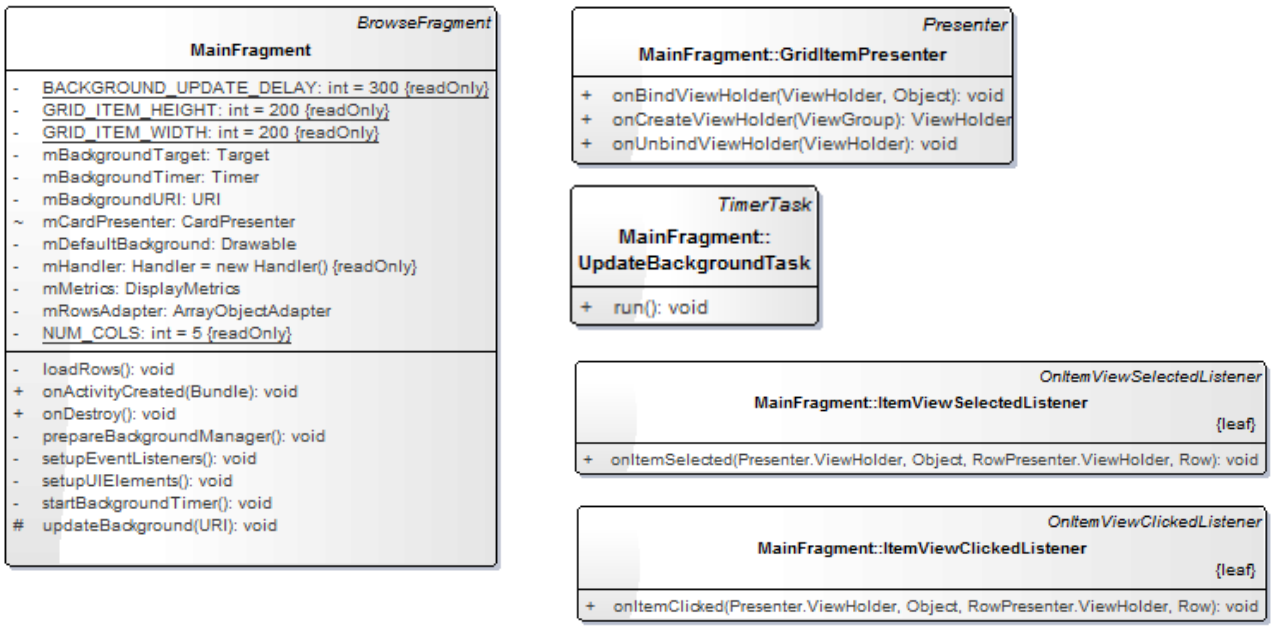


*Ilustración 24. Diagramas referentes al fragment principal.*

La ilustración 24 muestra los diagramas referentes al *fragment* principal, que contiene el catálogo. Se compone de una clase java para el propio *fragment*, con sus atributos y métodos para la carga del catálogo, aspectos gráficos y todo lo relativo al cambio de *background*. Incluye además, un *presenter* llamado *GridItemPresenter* para mostrar las preferencias en vista tipo *grid* o rejilla, un *timer* para el cambio de *background*, y dos *listeners* para la selección de los elementos del catálogo.

### *Fragment* de detalles

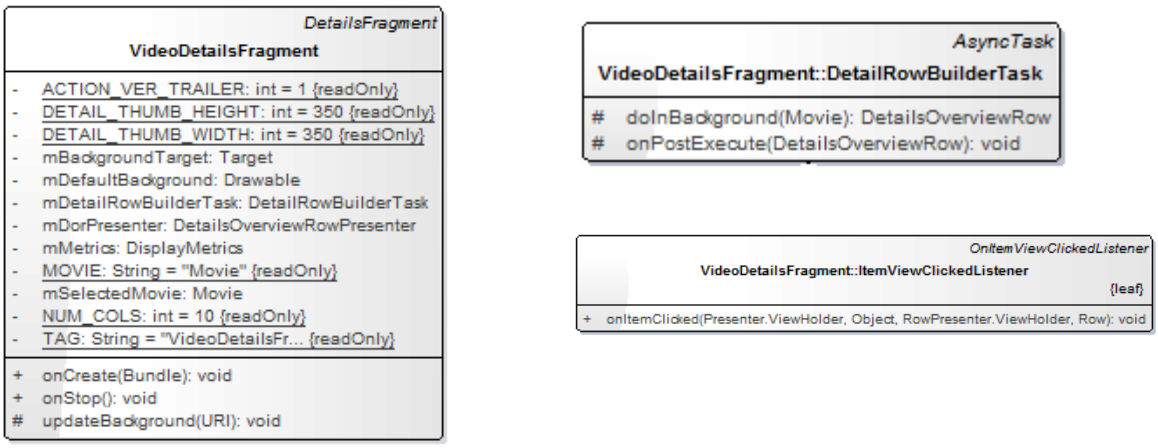


*Ilustración 25. Diagramas referentes a la vista de detalles.*

La ilustración 25 muestra los diagramas del *fragment* de la vista de detalles de los elementos. Se compone de una clase java para el propio *fragment,* una tarea asíncrona para la recepción de las imágenes de portada tipo póster procedentes de internet y un *listener* para la selección de elementos.

### *Fragment* de búsqueda

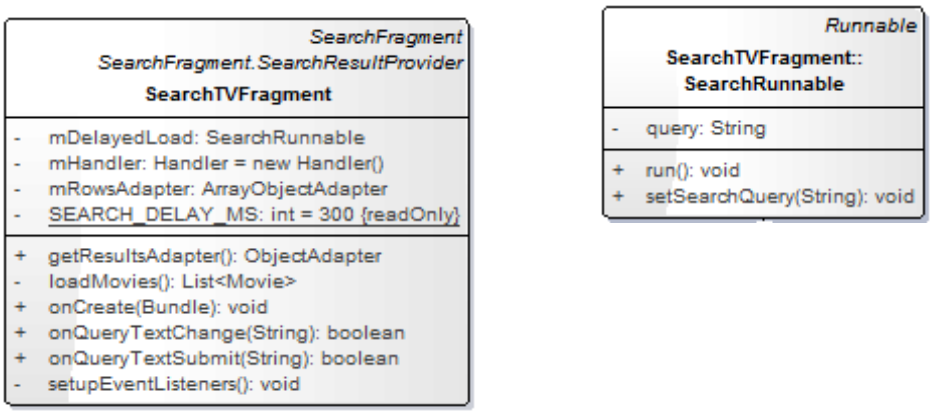


*Ilustración 26. Diagramas referentes al fragment de búsqeda*

La ilustración 26 muestra los diagramas referentes al *fragment* de búsqueda. Está compuesto por una clase java para el propio *fragment* con sus atributos y métodos para la obtención del texto introducido en el cuadro de búsqueda. Incluye un objeto *runnable* que se ejecuta cuando se obtiene texto y se encarga de realizar la búsqueda.

### *Fragment* de vista de rejilla

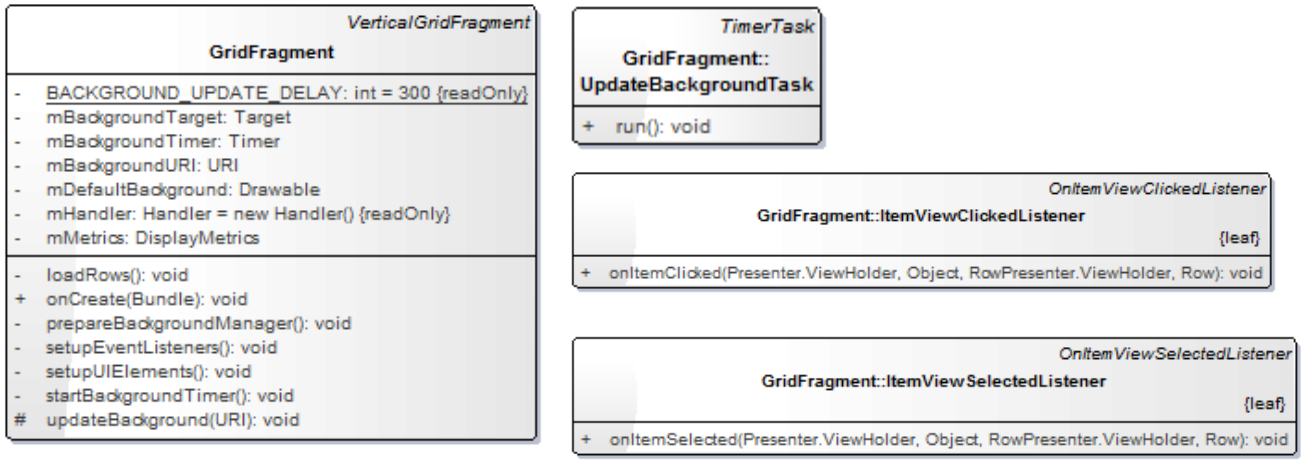


*Ilustración 27. Diagramas referentes al fragment de vista grid o rejilla.*

La ilustración 27 muestra los diagramas referentes al *fragment* de la vista de rejilla o *grid.* Está compuesto por una clase java, con sus atributos y métodos para la carga de elementos y todo lo relacionado con aspectos gráficos y cambio de *background.* Incluye dos *listeners* para la selección de los elementos y un *timer* utilizado para el cambio de *background.*

### Presenter tipo tarjeta o panel

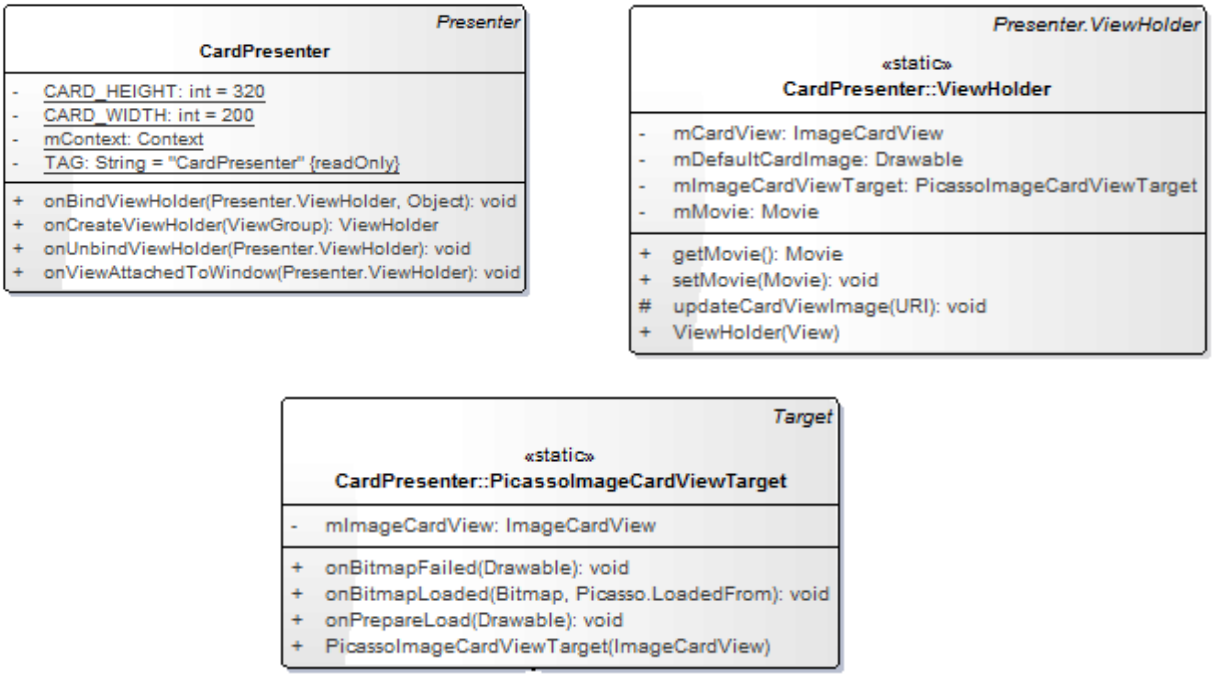


*Ilustración 28. Diagramas referentes al presenter utilizado para la visualización tipo tarjeta o panel de los elementos.*

La ilustración 28 muestra los diagramas referentes al *presenter* para la visualización de los elementos en tarjetas o paneles. Además de una clase java para propio *presenter,* se compone de una clase estática para la obtención de la información de cada elemento *Movie* para su posterior visualización en una tarjeta o panel y otra clase estática para la obtención de la imagen de internet y su acoplamiento en el *presenter.*

### *Presenter* vista de detalles

La ilustración 29 muestra el diagrama del *presenter* utilizado para la visualización de la vista de detalles. Contiene un solo método que obtiene la información necesaria de los elementos para mostrarla mediante un *AbstractDetailsDescriptionPresenter.*


*Ilustración 29. Diagrama del presenter de la vista de detalles.*

**Clase *Utils***

La ilustración 30 muestra el diagrama de la clase *Utils,* que contiene una serie de métodos comunes en varias clases.

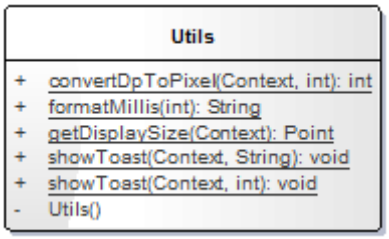


*Ilustración 30. Diagrama clase Utils.*

## *4.4 Desarrollo de la aplicación.*

En esta sección se va a explicar el desarrollo de lo que sería una aplicación típica para un televisor, en este caso, basada en Android TV. La aplicación se ha desarrollado partiendo de un código base proporcionado por Google. Se trata de un código en el que se asientan muchos de los principios básicos para el desarrollo en esta plataforma, como es el uso de *fragments*, *adapters* y *presenters*.

Se trata de una aplicación que contiene un listado de películas divididas en varias categorías, dispuestas en forma de catálogo. Al seleccionar alguna de las películas, se podrá visualizar información sobre ella, una pequeña sinopsis y una opción para visualizar el tráiler. Además, se podrán realizar búsquedas por medio del título. Como se puede apreciar, es un tipo de aplicación muy característica para visualizar en un televisor, ya que está orientada al consumo de contenido multimedia.

### 4.4.1 Código base y modificaciones y/o nuevas funcionalidades.

Como se ha comentado, la aplicación se construye en base a código proporcionado por Google. En este apartado se va a explicar qué funcionalidades proporciona dicho código, aquellas que se han modificado para adaptarlas al objetivo de la aplicación y nuevas funcionalidades implementadas.

**Pantalla principal – catálogo**

El código proporcionado muestra un catálogo formado por 6 categorías, cada una de ellas contiene 15 elementos, aunque en realidad sólo hay 5 elementos diferentes y se distribuyen por las 6 categorías colocándose aleatoriamente como se muestra en la ilustración 31. En este caso se ha modificado dicho catálogo para que exista un mapeo real entre las diferentes categorías y los elementos que contiene cada una, ya que

cada categoría debe contener unos elementos específicos. Para este caso, serán los diferentes géneros cinematográficos, que contendrán películas de cada género.

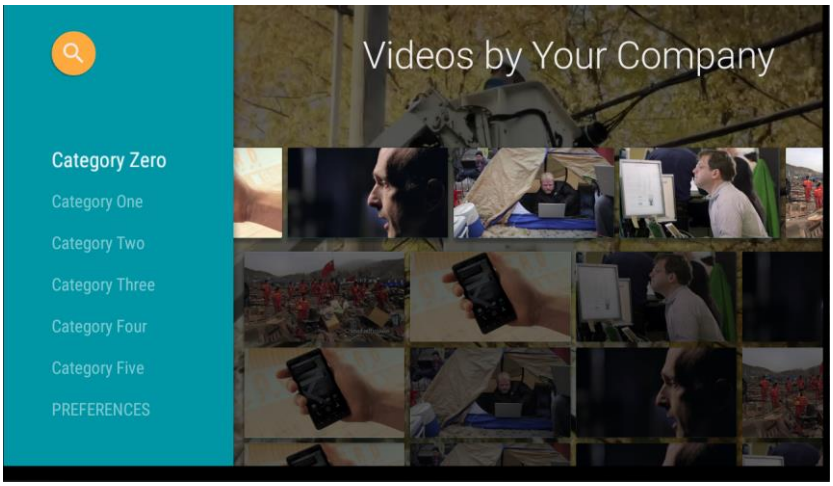


*Ilustración 31. Catálogo proporcionado por Google.*

Viene definido un *presenter* para la visualización de cada elemento tipo tarjeta o panel y el cambio de imagen de fondo acorde al elemento seleccionado como se aprecia en la ilustración 32. Aquí no se ha realizado ninguna modificación.


*Ilustración 32. Visualización tipo tarjetade cada elemento.*

Una de las categorías está pensada para las diferentes opciones o preferencias que podría tener la aplicación como se aprecia en la ilustración 33.

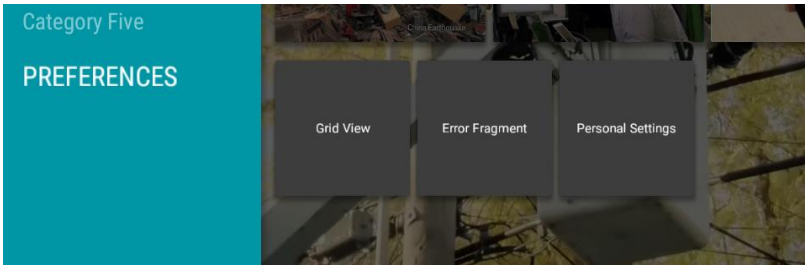


*Ilustración 33. Preferencias.*

La única opción que tenía funcionalidad es la "Error *Fragment*" que define una pantalla de error que se mostraría al usuario en caso de suceder algún tipo de error, aunque no

está implementada esta funcionalidad. En este caso, se ha eliminado dicha opción y en la opción *Grid View* se ha implementado una nueva vista tipo *grid* o rejilla de todos los elementos que contenga el catálogo mediante el uso de un nuevo *fragment*.

El icono de búsqueda no tenía ninguna funcionalidad. En este caso se ha implementado la funcionalidad que corresponde, es decir, un buscador por título de película que muestra los resultados en caso de producirse.

**Vista de detalles**

Esta *activity* apenas se ha modificado, define el *presenter* **AbstractDetailsDescriptionPresenter** y el *fragment* **DetailsFragment** de la librería leanback v17 de Android para construir la vista de detalles de cada película, como se aprecia en la ilustración 34.

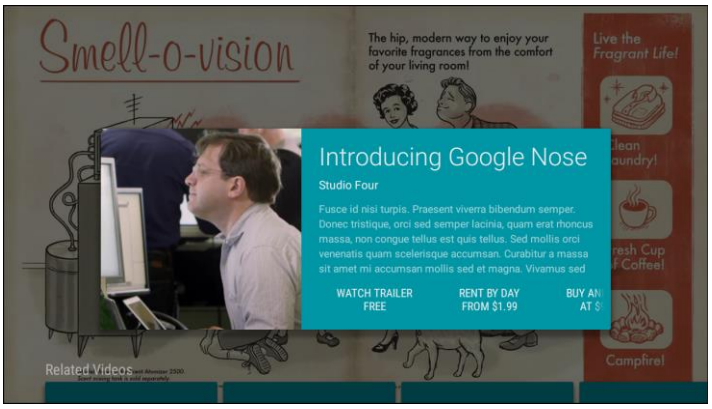


*Ilustración 34. Vista detalles.*

En este caso se han hecho modificaciones mínimas, se han modificado aspectos gráficos y se ha reducido el número de acciones posibles a 1, que se corresponde con "ver tráiler", ya que las dos opciones restantes dealquiler o compra de elemento no están implementados.

Además, proporciona un *fragment* inferior para mostrar un listado de contenido a modo de "Vídeos relacionados" que muestra los 5 elementos diferentes aleatoriamente ordenados durante 10 columnas, como se aprecia en la ilustración 35.

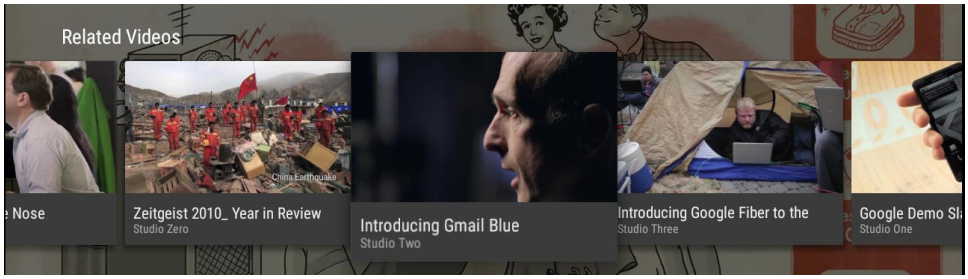


*Ilustración 35. Vídeos relacionados.*

En este caso se ha modificado para que sólo se muestren los elementos que pertenezcan a la misma categoría del elemento seleccionado en la vista de detalles.

**Vista de reproductor**

Esta *activity* sólo incluye un reproductor para la visualización de los vídeos. Igual que la vista de detalles, contiene, en la parte inferior, el listado de "Vídeos relacionados".

Esta *activity* apenas se modificará, ya que sólo está pensada para reproducir vídeos y no es necesario implementar un nuevo reproductor. Se ha eliminado el *fragment* inferior de elementos relacionados para hacer lo más liviano posible el reproductor, ya que la reproducción de contenido desde fuentes remotas, como internet, consume muchos recursos y el emulador es bastante limitado en este sentido. Esta parte se comentará más en detalle en el capítulo 5.

### 4.4.2 Obtención de contenido.

Antes de comenzar con las diferentes pantallas que conforman la aplicación, se va a explicar cómo se define y cómo se crea cada objeto *Movie* que representa cada película.

Una aplicación de estas características está pensada para consumir contenido multimedia en línea. Hoy en día existen multitud de repositorios de vídeos e imágenes y una aplicación de este estilo sería muy apropiada para su visualización en un dispositivo como es la televisión. En este caso, no se dispone de ningún repositorio ni servidor web propio para obtener los elementos multimedia, y, por otra parte, al tratarse de un estudio e introducción a la programación en Android TV, los elementos multimedia se van a construir de forma local, recibiendo de la web algunos aspecto como son las diferentes imágenes y URLs de cada película.

Para ello se ha definido una clase **Movie.java,** donde se definen los diferentes atributos que va a tener cada película y los habituales métodos *get* y *set*. En este caso, cada película va a tener los siguientes atributos:

- Categoría: define la categoría a la que pertenece la película.
- Título: el título de la película.
- Descripción: sinopsis de la película.
- Imagen de portada: define la URL de dónde se carga la imagen de portada.
- Imagen de fondo: define la URL de dónde se carga la imagen de fondo.
- Vídeo: define la URL de donde se carga el vídeo, que contiene el tráiler de la película.

Por otra parte, la clase **MovieList.java** contiene los métodos para generar el listado de películas de cada una de las categorías. Con la información que se proporcione en esta clase, primero se generan los diferentes objetos tipo *Movie,* que representan cada una de las películas con sus atributos comentados anteriormente. Después, todos los objetos Movie que tengan la misma categoría se añaden a una misma lista tipo *List<Movie>.* Con ello, se obtienen tantas listas como categorías diferentes describamos.

### 4.4.3 Pantallas de la aplicación.

Una vez se obtienen todos los listados de películas, el siguiente paso es integrarlos en el catálogo mediante el uso de *adapters* y *presenters.*

#### Pantalla principal

Como se comentó anteriormente, el *fragment* principal es del tipo **BrowseFragment**, definido en la librería Leanback v17 para Android TV, y está especialmente pensado para definir catálogos de una manera muy sencilla y orientada totalmente para la televisión.

Un BrowseFragment está compuesto por dos *fragments*, un HeadersFragment, que compone el *fragment* izquierdo y contiene las diferentes categorías, y un RowsFragment, que compone el *fragment* derecho y muestra los elementos de las categorías del *fragment* anterior, en este caso, las diferentes películas.

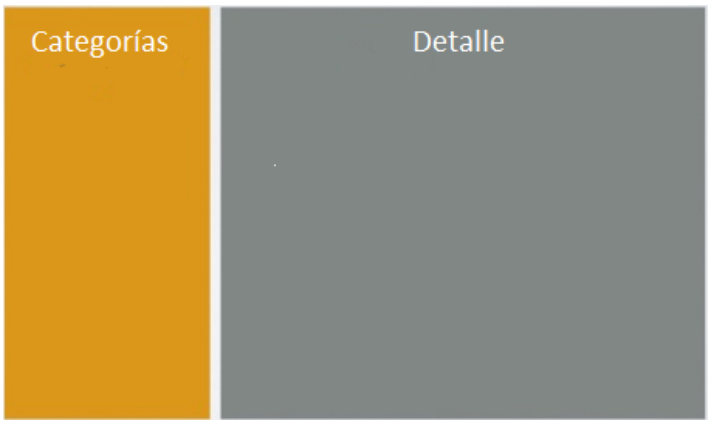


*Ilustración 36. Ejemplo de BrowseFragment.*

En el caso del HeadersFragment, para que sea visible hay que utilizar el método llamado *setHeadersState(int estado)* y como parámetro una de las tres siguientes constantes:

| | |
|---|---|
| HEADERS_DISABLED | El HeadersFragment está deshabilitado y no se muestra. |
| HEADERS_ENABLED | El HeadersFragment está habilitado y se muestra por defecto. |
| HEADERS_HIDDEN | El HeadersFragment está habilitado, pero no se muestra por defecto. |

*Tabla 3. Estados posibles para la visualización de los elementos de HeadersFragment.*

Por otra parte, para regresar al HeadersFragment una vez se esté navegando por el RowsFragment pulsando el botón *Back,* se utiliza, por defecto, el método *setHeadersTransitionOnBackEnabled(boolean),* con parámetro *true,* siempre que el estado del HeadersFragment sea HEADERS_ENABLED o HEADERS_HIDDEN. Si esta acción la quiere controlar el programador por sí mismo, se debe utilizar el parámetro *false,* e implementar su propio método en la clase *BrowseFragment.BrowseTransitionListener*.

Para editar la apariencia del BrowseFragment existen varios métodos, los más característicos se describen a continuación:

- Para cambiar el título, mostrado en la parte superior derecha de la pantalla, existen dos métodos: *setTitle(String titulo),* como parámetro recibe una cadena de texto con dicho título, o *setBadgeDrawable(Drawable drawable),* que recibe un objeto tipo *Drawable,* que normalmente se trata de una imagen.
- Para cambiar el color principal de la aplicación, como es el color de fondo del HeadersFragment o el color de fondo de los elementos mostrados en el RowsFragment, se utiliza el método setBrandColor(int color), que recibe como parámetro un número entero que representa un color, habitualmente definido en el fichero *colors.xml* de un proyecto Android:

```
setBrandColor(getResources().getColor(R.color.color_principal));
```

Y en el fichero colors.xml:

```
<color name=" color_principal ">#2CB254</color>
```

Siendo #2CB254, el código HTML del color, en este caso, un tipo de verde.

Una vez comentadas las características básicas del BrowseFragment, se va a proceder a explicar cómo se rellena dicho *fragment* con los datos y elementos deseados, en este caso, un catálogo de películas, con las diferentes categorías en el HeadersFragment y cada una de las películas en el RowsFragment, dentro de su categoría correspondiente.

Para ello se ha implementado un método llamado *loadRows()*, que lleva a cabo todas las acciones pertinentes.

```
private void loadRows() {
List<Movie> listAventuras = MovieList.setupMoviesAventuras();
List<Movie> listAccion = MovieList.setupMoviesAccion();
List<Movie> listComedia = MovieList.setupMoviesComedia();
List<Movie> listAnimacion = MovieList.setupMoviesAnimacion();

mRowsAdapter = new ArrayObjectAdapter(new ListRowPresenter());
mCardPresenter = new CardPresenter();

ArrayObjectAdapter listRowAdapter = new
ArrayObjectAdapter(mCardPresenter);
            for (int j = 0; j < NUM_COLS; j++) {
                listRowAdapter.add(listAventuras);
            }

ArrayObjectAdapter listRowAdapterAccion = new
ArrayObjectAdapter(mCardPresenter);
            for (int j = 0; j < NUM_COLS; j++) {
                listRowAdapterAccion.add(listAccion);
            }

ArrayObjectAdapter listRowAdapterComedia = new
ArrayObjectAdapter(mCardPresenter);
            for (int j = 0; j < NUM_COLS; j++) {
                listRowAdapterComedia.add(listComedia);
            }

ArrayObjectAdapter listRowAdapterAnimacion = new
ArrayObjectAdapter(mCardPresenter);
            for (int j = 0; j < NUM_COLS; j++) {
                listRowAdapterAnimacion.add(listCienciaAnimacion.get(j % 5));
            }
```

En esta primera parte del método se guardan las listas de cada categoría con todas las películas correspondientes en objetos tipo *List<Movie>* y se rellena un adapter de tipo ArrayObjectAdapter, de la librería Leanback v17, con dichas películas, por cada categoría. La constante NUM_COLS define el número de columnas que tendrá el RowsFragment, en este caso un total de cinco.

Cabe destacar que cada adapter para las películas recibe como parámetro un objeto tipo *Presenter*. Una vez finalizada la explicación del mapeo entre el HeadersFragment y el RowsFragment, se abordará detalladamente todo lo referente al tema de los presenters.

El método finaliza de la siguiente forma:

```
HeaderItem header = new HeaderItem(0, MovieList.MOVIE_CATEGORY[0], null);
HeaderItem headerAccion = new HeaderItem(1, MovieList.MOVIE_CATEGORY[1],
null);
HeaderItem headerComedia = new HeaderItem(2, MovieList.MOVIE_CATEGORY[2],
null);
HeaderItem headerAnimacion = new HeaderItem(3, MovieList.MOVIE_CATEGORY[3],
null);

mRowsAdapter.add(new ListRow(header, listRowAdapter));
mRowsAdapter.add(new ListRow(headerAccion, listRowAdapterAccion));
mRowsAdapter.add(new ListRow(headerComedia, listRowAdapterComedia));
mRowsAdapter.add(new ListRow(headerAnimacion, listRowAdapterAnimacion));
```

```
  setAdapter(mRowsAdapter);

    }
```

Se define un objeto tipo *HeaderItem* para cada categoría, que se corresponde con cada elemento mostrado en el HeadersFragment, y se asocia con el adapter correspondiente comentados anteriormente. Para facilitar la explicación:

```
HeaderItem headerAnimacion = new HeaderItem(3, MovieList.MOVIE_CATEGORY[3],
null);
```

Este HeaderItem se corresponde con la categoría Animación, el primer parámetro es la posición que ocupará en el HeadersFragment, en este caso la cuarta posición, ya que la numeración comienza en el cero y recibe como parámetro el número tres. El segundo parámetro indica el texto que aparecerá como nombre de la categoría. En la clase *MovieList,* viene definido de la siguiente manera, por lo que su nombre será "Animación":

```
public static final String MOVIE_CATEGORY[] = {
            "Aventura",
            "Acción",
            "Comedia",
            "Animación"
    };
```

Con esto, quedan configurados cada uno de los elementos HeaderItem y, por tanto, sólo queda incluir dichos objetos en el HeadersFragment y asociarlos a la lista de películas que pertenezca a la misma categoría. Para ello se utiliza un adapter global para el BrowseFragment, en este caso **mRowsAdapter,** con la particularidad de que el presenter recibido como parámtero tiene que ser de tipo Row, para generar un listado vertical donde cada fila de la lista se corresponde con cada HeaderItem.

Por ello se ha utilizado ListRowPresenter:

```
mRowsAdapter = new ArrayObjectAdapter(new ListRowPresenter());
```

Las filas de dicho adapter se definen como un objeto *ListRow,* y como parámetros se le indica el HeaderItem y el adapter que contiene las películas de cada categoría:

```
mRowsAdapter.add(new ListRow(headerAnimacion, listRowAdapterAnimacion));
```

Para finalizar, una vez configurado el adapter global del BrowseFragment, se invoca el método **setAdapter(adapter),** para establecerlo:

```
setAdapter(mRowsAdapter);
```

Resultado obtenido:

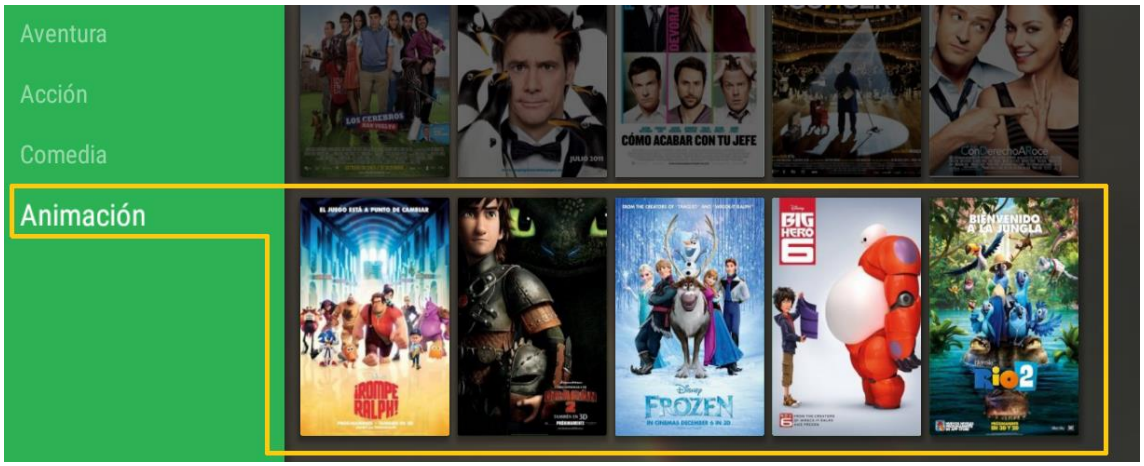


*Ilustración 37. Correspondencia de cada categoría con sus películas asociadas.*

Una vez llegado a este punto, se va a proceder a explicar la funcionalidad de los *Presenters,* comentados anteriormente, para la visualización de cada una de las películas en un formato tipo "tarjeta", como se aprecia en la ilustración superior. Para facilitar la explicación, se muestra de nuevo el fragmento de código en el que el adapter con las películas de cada categoría recibe como parámetro un objeto tipo presenter (en este caso, el adapter de la categoría de animación):

```
ArrayObjectAdapter listRowAdapterAnimacion = new ArrayObjectAdapter(mCardPresenter);
```

De este modo, se indica, mediante dicho presenter, cómo se van a visualizar los diferentes elementos que contenga el adapter, en este caso **mCardPresenter,** es un presenter de tipo CardPresenter, definido en la clase **CardPresenter.java**.

Para construir el presenter, la clase donde se defina debe extender la clase Android *Presenter*, perteneciente a la librería leanback v17:

```
public class CardPresenter extends Presenter {…}
```

En esta clase hay que sobrescribir el método **onCreateViewHolder**, en el que se define el tipo de objeto *View* utilizado para visualizar los elementos del presenter. Un objeto View de andorid, como se comentó en el capítulo de estado del arte, actúa como un contenedor para un elemento de la interfaz de usuario. En este caso se ha utilizado el tipo **ImageCardView**, que está pensado para contener un objeto cuyo principal elemento sea una imagen. Este ImageCardView se ajusta perfectamente a las necesidades del catálogo, ya que el elemento principal de cada "tarjeta" es la portada de la película, con un pequeño texto en la parte inferior para el título.

En este mismo método también se configuran aspectos visuales y funcionalidades del propio ImageCardView. En este caso se han habilitado dos métodos, *setFocusable* y *setFocusableInTouchMode* y se ha modificado el color de fondo:

```
ImageCardView cardView = new ImageCardView(mContext);
cardView.setFocusable(true);
cardView.setFocusableInTouchMode(true);
cardView.setBackgroundColor(mContext.getResources().getColor(R.color.fondo_tar
jeta));
```

Los dos primeros métodos, setFocusable y setFocusableInTouchMode, con parámetro *true*, habilitan que el objeto ImageCardView resalte cuando se haya seleccionado y destaque sobre el resto. Con ello se facilita la navegación con un mando del tipo D-Pad, ya que, en todo momento, se puede visualizar el elemento seleccionado:

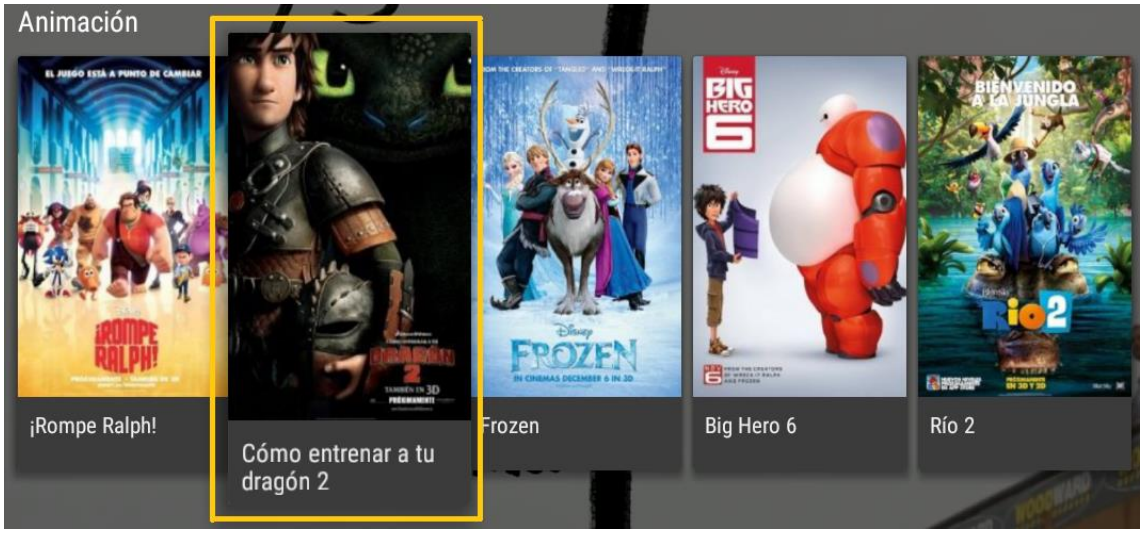


*Ilustración 38. Ejemplo de un elemento con setFocusable habilitado.*

Ambos métodos realizan la misma función, pero en el caso de setFocusableInTouchMode, va orientado a dispositivos con entrada táctil. Por su parte, el método *setBackgroundColor*, establece un color de fondo.

Una vez se configuran las opciones visuales del presenter tipo "tarjeta", se ejecuta el método **onBindViewHolder**, que se encarga de enlazar el objeto devuelto por el anterior método, con el elemento que queremos visualizar dentro de dicho presenter, en este caso, una película. Como se aprecia en la ilustración anterior, sólo se necesita recoger el título y la imagen de portada:

En esta primera parte se define el objeto a enlazar con el presenter, un objeto tipo *Movie*.

```
public void onBindViewHolder(Presenter.ViewHolder viewHolder, Object item) {
        Movie movie = (Movie) item;
        ((ViewHolder) viewHolder).setMovie(movie);
```

```
if (movie.getCardImageUrl() != null) {
 ((ViewHolder)  viewHolder).mCardView.setTitleText(movie.getTitle());
 ((ViewHolder) viewHolder).mCardView.setMainImageDimensions(CARD_WIDTH,
CARD_HEIGHT);
 ((ViewHolder) viewHolder).updateCardViewImage(movie.getCardImageURI());
        }
    }
```

Y para finalizar, con el método **setTitleText** se obtiene el título de la película accediendo al atributo Title, mediante el método getTitle(). Se obtiene de la misma forma la URL de la imagen de portada, mediante el método **updateCardViewImage** y se establecen las dimensiones, alto y ancho, de la tarjeta mediante **setMainImageDimensions.**

En el caso del método **updateCardViewImage,** hace uso de una librería Android llamada *Picasso*. Se trata de una librería para facilitar la obtención y procesado de imágenes procedentes de fuentes externas, como en este caso, ya que se obtienen de internet.

```
protected void updateCardViewImage(URI uri) {
            Picasso.with(mContext)
                    .load(uri.toString())
                    .resize(Utils.convertDpToPixel(mContext, CARD_WIDTH),
                            Utils.convertDpToPixel(mContext, CARD_HEIGHT))
                    .error(mDefaultCardImage)
                    .into(mImageCardViewTarget);
        }
```

Como parámetro recibe una URI, en este caso, la URL de la imagen. Con el método *load,* se obtiene dicha uri y se transforma en cadena de texto, el método *resize* fija las dimensiones de dicha imagen. Para ello utiliza el método *convertDpToPixel,* de la clase Utils, que se encarga de transformar las unidades a pixels. En el método *error* se define una imagen por defecto en caso de que se produzca algún error a la hora de obtener la imagen de internet. Por último el método *into* define el objeto tipo *View* donde se almacena la información.

A modo de contenedor para los diferentes ajustes que podría tener la aplicación, el código de Google añade un elemento llamado PREFERENCES. En este caso, no va a contener películas, por lo que no se usará el presenter de tipo tarjeta o panel, se define un sencillo presenter para mostrar las diferentes opciones.

```
private class GridItemPresenter extends Presenter {
        @Override
        public ViewHolder onCreateViewHolder(ViewGroup parent) {
            TextView view = new TextView(parent.getContext());
            view.setLayoutParams(new ViewGroup.LayoutParams(GRID_ITEM_WIDTH,
GRID_ITEM_HEIGHT));
            view.setFocusable(true);
            view.setFocusableInTouchMode(true);

view.setBackgroundColor(getResources().getColor(R.color.default_background));
            view.setTextColor(Color.WHITE);
            view.setGravity(Gravity.CENTER);
```

```
            return new ViewHolder(view);
        }
```

Es un presenter muy sencillo, que sólo contiene un campo de texto, *TextView*, que contendrá el texto de cada opción, se le fijan unas dimensiones con el método *setLayoutParams,* se habilita el resaltado del elemento, como se comentó anteriormente con *setFocusable* y *setFocusableInTouchMode,* se aplica un color de fondo y color y posición del texto.

Para añadirlo al catálogo, se incluye el siguiente código en el mismo método **loadRows()** que rellenaba todo el contenido con las películas y categorías.

```
HeaderItem gridHeader = new HeaderItem(4,
getResources().getString(R.string.preferences), null);

        GridItemPresenter mGridPresenter = new GridItemPresenter();
        ArrayObjectAdapter gridRowAdapter = new
ArrayObjectAdapter(mGridPresenter);
        gridRowAdapter.add(getResources().getString(R.string.grid_view));

gridRowAdapter.add(getResources().getString(R.string.personal_settings));
        mRowsAdapter.add(new ListRow(gridHeader, gridRowAdapter));
```

Se crea un nuevo *HeaderItem,* con el texto “Preferencias” y un *adapter* que hace uso del *presenter* comentado anteriormente. A dicho *adapter* se le agregan dos opciones: una denominada “Grid View” y otra “Opciones personales”. Nótese que se ha eliminado la opción de Error Fragment, del código original de Google, ya que no tiene funcionalidad definida.

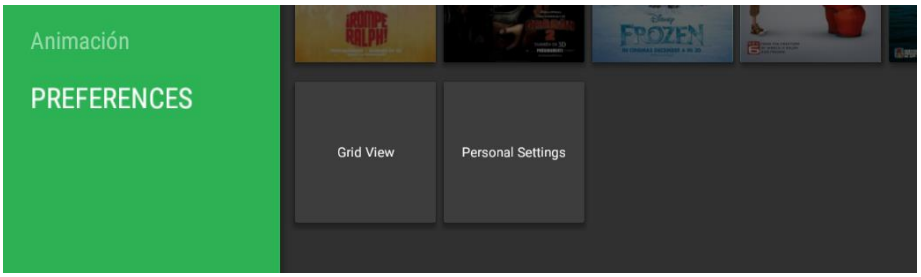


*Ilustración 39. Visualización de opciones.*

Por otra parte, cabe destacar la actualización continua de la imagen de fondo de la aplicación. Como se comentó anteriormente, es un método muy útil para mejorar el aspecto gráfico y mejorar la interactividad con el usuario. En este caso, cuando el usuario se posicione sobre alguna de las películas, se cargará una imagen de fondo asociada a dicha película.

Estos cambios de fondo se consiguen gracias a la clase **BackgourndManager,** de la librería leanback v17. En cada *activity* donde se pretenda realizar cambios en la imagen

de fondo, hay que definir un **BackgourndManager** y fijarlo a dicha *activity* mediante el método *attach.*

```
BackgroundManager backgroundManager =
BackgroundManager.getInstance(getActivity());
        backgroundManager.attach(getActivity().getWindow());

mDefaultBackground =
getResources().getDrawable(R.drawable.default_background);
```

En este fragmento de código se llama al método mencionado antes y se define una imagen de fondo por defecto.

Para actualizar la imagen de fondo, se define un *Timer*, que comprueba cada cierto tiempo si se debe cambiar el fondo. Dicho tiempo viene definido por una variable global llamada BACKGROUND_UPDATE_DELAY, en este caso, 300 ms.

```
private void startBackgroundTimer() {
        if (null != mBackgroundTimer) {
            mBackgroundTimer.cancel();
        }
        mBackgroundTimer = new Timer();
        mBackgroundTimer.schedule(new UpdateBackgroundTask(),
BACKGROUND_UPDATE_DELAY);
    }
```

Para ello, cada 300 ms se ejecuta una función llamada **UpdateBackgroundTask()** que comprueba si la variable **mBackgroundURI** ha sido modificada.

```
private class UpdateBackgroundTask extends TimerTask {

        @Override
        public void run() {
            mHandler.post(new Runnable() {
                @Override
                public void run() {
                    if (mBackgroundURI != null) {
                        updateBackground(mBackgroundURI);
                    }
                }
            });

        }
    }
```

En caso positivo, mediante otro método, **updateBackground(),** haciendo uso, de nuevo, de la librería *Picasso,* define la nueva imagen de fondo de la *activity*.

Una vez llegado a este punto, se ha obtenido un catálogo dividido en varias categorías mediante el uso de un adapter y un presenter y la visualización de las películas en objetos tipo “tarjeta” mediante otro presenter y las opciones. El siguiente paso es definir qué acción se va a llevar a cabo cuando se seleccione alguna película u opción del catálogo.

En el caso de una película, se ha definido una nueva *activity* donde se visualizarán detalles concretos de cada película, como una sinopsis y el acceso al tráiler.

Para ello se hace uso de un método de la clase BrowseFragment llamado ***setOnItemViewClickedListener(OnItemViewClickedListener listener)***, que recibe como parámetro un objeto listener. Dicho listener se encarga de capturar la acción de que un usuario haya accedido a una película y define cómo actuar en consecuencia.

Definición del listener:

```
private final class ItemViewClickedListener implements
OnItemViewClickedListener {

public void onItemClicked(Presenter.ViewHolder itemViewHolder, Object item,
                          RowPresenter.ViewHolder rowViewHolder, Row row) {

   if (item instanceof Movie) {
         Movie movie = (Movie) item;
         Intent intent = new Intent(getActivity(), DetailsActivity.class);
         intent.putExtra(DetailsActivity.MOVIE, movie);
        getActivity().startActivity(intent);
         } else if (item instanceof String) {
                if (((String) item).contains("Grid")) {
                    Intent intent = new Intent(getActivity(),
GridActivity.class);
                    startActivity(intent);
                } else {
                    Toast.makeText(getActivity(), (String) item,
Toast.LENGTH_SHORT).show();
                }
     }
}
```

En dicho listener se comprueba si el elemento que se ha seleccionado es de tipo *Movie,* es decir, se está accediendo a una película. De ser así, se genera un nuevo ***Intent* ,** que se encarga de ejecutar la nueva *activity* a mostrar al usuario. Para ello se indica qué *activity* se a mostrar, en este caso, la vista de detalles de la película, **DetailsActivity,** y se le pasa un parámetro extra, indicando el objeto Movie que se ha seleccionado para poder acceder y visualizar la información de dicha película en la nueva *activity*.

En caso de que el elemento seleccionado sea de tipo *String,* una cadena de texto, es que se está accediendo a una de las opciones. De ser así, se comprueba si se accede a la opción “Grid View” para mostrar una nueva vista de las películas mediante el uso de un *presenter* diferente. La opción de “Opciones personales” no tiene ninguna funcionalidad definida.

### Pantalla con vista de detalles

Una vez se accede a un elemento del catálogo, como se acaba de explicar, se lanzará una nueva *activity* donde se visualiza contenido detallado de dicho elemento. Esta

*activity* tiene asociado un layout, **activity_details.xml**, en el que se define un nuevo *fragment* donde se visualizará el contenido de dicha *activity*, **VideoDetailsFragment**.

Del mismo modo que en la *activity* principal se utilizó un *fragment* de tipo BrowseFragment, precisamente pensado para la visualización de catálogos, para la vista de detalles será necesario el uso de un *fragment* diferente. Para la visualización de vistas de detalle, Android TV proporciona el *fragment* **DetailsFragment,** el *fragment* de detalles será de este tipo**.** La idea es mostrar el título, un pequeño texto a modo de sinopsis, acompañado de la imagen de portada y un acceso al tráiler de la película y para ello, la librería Leanback v17 de Android TV cuenta con el presenter **DetailsOverviewRowPresenter.**


*Ilustración 40. Esquema de un presenter DetailsOverviewRowPresenter.*

Para visualizar el contenido de cada elemento dentro del presenter, hay que obtener la información necesaria de cada película. En el caso de la descripción o sinopsis y el título, se obtienen mediante un segundo presenter que se debe pasar por parámetro al primero:

```
mDorPresenter =
   new DetailsOverviewRowPresenter(new DetailsDescriptionPresenter());
```

Este nuevo presenter, **DetailsDescriptionPresenter**, extiende de la clase *AbstractDetailsDescriptionPresenter,* especialmente pensada para el renderizado de textos, que conforman la descripción de un elemento en la vista de detalles. Es un presenter muy sencillo, que tan sólo obtiene el título y sinopsis:

```
        public class DetailsDescriptionPresenter extends
AbstractDetailsDescriptionPresenter {

    protected void onBindDescription(ViewHolder viewHolder, Object item) {
        Movie movie = (Movie) item;

        if (movie != null) {
            viewHolder.getTitle().setText(movie.getTitle());
```

```
            viewHolder.getBody().setText(movie.getDescription());
        }
    }
}
```

El texto que devuelva el método getTitle se visualizará en el apartado de título y el texto devuelto por getBody en el apartado de la descripción de la ilustración 8.

Para la imagen y las diferentes acciones, se debe definir un objeto tipo ***DetailsOverviewRow*** y su obtención se lleva cabo mediante el uso de una tarea asíncrona de Android. Para ello se utiliza la clase abstracta *AsyncTask,* que permite llevar a cabo diversas tareas en segundo plano y visualizar sus resultados directamente en la interfaz de usuario, sin necesidad de definir otros hilos de ejecución o subprocesos.

```
private class DetailRowBuilderTask extends AsyncTask<Movie, Integer,
DetailsOverviewRow> {
        @Override
protected DetailsOverviewRow doInBackground(Movie... movies) {
mSelectedMovie = movies[0];
DetailsOverviewRow row = new DetailsOverviewRow(mSelectedMovie);
  try {
    Bitmap poster = Picasso.with(getActivity())
                        .load(mSelectedMovie.getCardImageUrl())

.resize(Utils.convertDpToPixel(getActivity().getApplicationContext(),
DETAIL_THUMB_WIDTH),

Utils.convertDpToPixel(getActivity().getApplicationContext(),
DETAIL_THUMB_HEIGHT))
.centerCrop()
.get();
row.setImageBitmap(getActivity(), poster);
       } catch (IOException e) {
}
```

En este caso, la imagen se ha tratado como un objeto Bitmap (imagen de mapa de bits) en lugar de drawable. Funcionalmente no hay diferencias en este caso, un drawable, para Android, es cualquier objeto que se pueda “dibujar”, igual puede ser una imagen, un color, un layout, etc. En este caso concreto, el póster de la película será siempre una imagen.

Se ha recurrido de nuevo a la librería Picasso para obtención y tratamiento de imágenes. Igual que en el catálogo, se carga la URL con el método *load* y con el método *resize,* haciendo uso de la clase Utils, se fijan las dimensiones del póster. Por su parte, el método *centerCrop,* recorta parte de la imagen, arriba, abajo, derecha e izquierda y escala la parte restante para que ocupe el total de las dimensiones fijadas anteriormente y con el método *get,* se obtiene el resultado final.

Sólo falta añadir el póster al objeto *DetailsOverviewRow,* mediante el método **setImageBitmap** y las diferentes acciones que vayan a poder llevarse a cabo. En este caso, habrá una única acción, "Ver tráiler", pero se pueden añadir cuantas se quieran.

```
row.addAction(new Action(ACTION_VER_TRAILER, "Ver tráiler");
```

Una vez que finaliza el paso *doInBackground*, de la tarea asíncrona, y se ha obtenido el póster y generado la acción para ver el tráiler, se ejecuta el método *onPostExecute*, que es el paso final de dicha tarea.

```
protected void onPostExecute(DetailsOverviewRow detailRow) {
ClassPresenterSelector ps = new ClassPresenterSelector();

mDorPresenter.setBackgroundColor(getResources().getColor(R.color.detail_backgr
ound));
mDorPresenter.setStyleLarge(true);
mDorPresenter.setOnActionClickedListener(new OnActionClickedListener() {

                public void onActionClicked(Action action) {
                    if (action.getId() == ACTION_VER_TRAILER) {
                        Intent intent = new Intent(getActivity(),
PlaybackOverlayActivity.class);

intent.putExtra(getResources().getString(R.string.movie), mSelectedMovie);

intent.putExtra(getResources().getString(R.string.should_start), true);
startActivity(intent);
                    }
                }
            });
ps.addClassPresenter(DetailsOverviewRow.class, mDorPresenter);
ps.addClassPresenter(ListRow.class, new ListRowPresenter());
```

En este paso se configuran opciones visuales, en este caso, el color de fondo con el método *setBackgroundColor* y con el método *setStyleLarge*, con parámetro true, se habilita que el estilo del presenter sea de tamaño grande. De este modo se visualiza la sinopsis completa, por el contrario, si no se habilita sólo s emuestra la primera línea de texto de párrafo.Por último, se define cómo actuar al pulsar sobre la acción con identificador **ACTION_VER_TRAILER**, en este caso se abre una nueva *activity* con un reproductor para visualizar el tráiler. Como parámetro, se indica la película seleccionada para poder acceder a la URL de su tráiler correspondiente.


*Ilustración 41. Ejemplo de película en la vista de detalles mediante DetailsOverviewRowPresenter.*

En la parte inferior de la pantalla se ha creado una zona denominada “Vídeos relacionados”, donde se cargan las películas con la misma categoría, a modo de recomendaciones. Para ello se recurrido a un nuevo adapter y presenter, como los de la vista de catálogo, ya que se trata de una fila con varias “tarjetas” para mostrar las diferentes películas.

```
ArrayObjectAdapter listRowAdapter = new ArrayObjectAdapter(new
CardPresenter());
            for (int j = 0; j < NUM_COLS; j++) {
                listRowAdapter.add(list.get(j % 5));
            }

HeaderItem header = new HeaderItem(0, “Vídeos relacionados”,null);
            adapter.add(new ListRow(header, listRowAdapter));

setAdapter(adapter);
```

El código es igual que el utilizado a la hora de rellenar el BrowseFragment para generar el catálogo, pero con una sola categoría (“vídeos relacionados”) y el mismo presenter de tipo CardPresenter para la visualización tipo “tarjeta” o panel.

Cabe destacar, que al convivir dos presenters diferentes, se debe indicar cuál de ellos tiene que usar el sistema en función de la clase a la que pertenezca el componente a visualizar. Para ello, se define un objeto **ClassPresenterSelector**, que recibe por parámetro la clase java de dicho componente y el presenter que debe utilizar. Por ello, en el caso de la vista detallada utilizará el presenter *mDorPresenter*, ya que la clase asociada es *DetailsOverviewRow*, y, por su parte, los vídeos relacionados utilizarán el presenter *ListRowPresenter*, ya que su clase asociada es de tipo *ListRow.*

```
ClassPresenterSelector ps = new ClassPresenterSelector();

ps.addClassPresenter(DetailsOverviewRow.class, mDorPresenter);
ps.addClassPresenter(ListRow.class, new ListRowPresenter());
```

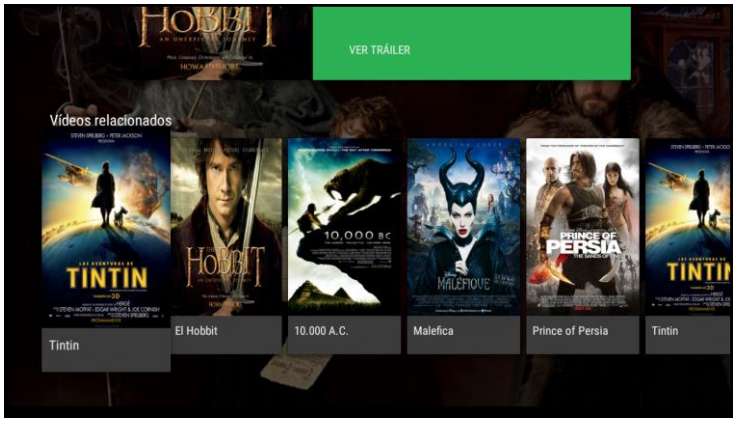


*Ilustración 42. Sección de vídeos relacionados en la parte inferior.*

Por último, comentar que el cambio de imagen de fondo se realiza del mismo modo que en la *activity* principal, mediante la clase **BackgroundManager.**

### Pantalla con vista de rejilla

En este apartado se va a explicar la *activity* que contiene la vista "Grid View", al seleccionar dicha opción en el apartado de preferencias.

Es una *activity* muy sencilla, pensada para probar el uso de un *fragment* diferente al **BrowseFragment** y **DetailsFragment,** utilizados a hasta ahora. Para ello se ha hecho uso de **VerticalGridFragment,** pensado para mostrar listados de elementos con una visualización tipo rejilla, junto con un nuevo *presenter* denominado **VerticalGridPresenter.**

```
public class GridFragment extends VerticalGridFragment {…}
```

Como se van a visualizar todas las películas, sin importar la categoría, primero se guardan todas ellas en una sola lista y con el método *shuffle,* se ordenan aleatoriamente.

```
private List<Movie> loadMovies() {

        ArrayObjectAdapter adapter = new ArrayObjectAdapter(new
CardPresenter());
        List<Movie> moviesAventuras = MovieList.listAventuras;
        List<Movie> moviesAnimacion = MovieList.listAnimacion;
        List<Movie> moviesComedia = MovieList.listComedia;
        List<Movie> moviesAccion = MovieList.listAccion;
        List<Movie> movies = new ArrayList<>();
        movies.addAll(moviesAventuras);
        movies.addAll(moviesAnimacion);
        movies.addAll(moviesComedia);
        movies.addAll(moviesAccion);

        Collections.shuffle(movies);
         adapter.addAll(0, movies);
        setAdapter(adapter);}
```

Cabe destacar que para la visualización de cada una de las películas se reutiliza el *presenter* de tipo tarjeta, **CardPresenter.**

En la creación del *fragment*, se define el **VerticalGridPresenter,** que será el encargado de establecer la visualización tipo rejilla de todos los elementos. En este caso se definen que tenga 4 columnas.

```
VerticalGridPresenter presenter = new VerticalGridPresenter();
        presenter.setNumberOfColumns(4);
        setGridPresenter(presenter);
```

Con todo ello, se tiene un **VerticalGridFragment** que contiene todas las películas en una visualización tipo rejilla mediante el **VerticalGridPresenter** y cada película se muestra mediante el **CardPrsenter**, reutilizado del catálogo.

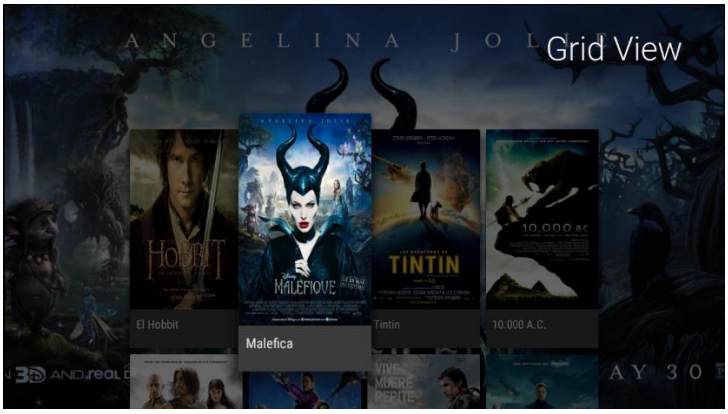


*Ilustración 43. Vista tipo rejilla mediante VerticalGridFragment.*

### Pantalla con reproductor

En el caso de esta *activity*, no se va a entrar en detalle, ya que se ha utilizado la *activity* de reproducción proporcionada por Google. Se trata de una *activity* sencilla, con un elemento **VideoView,** que actúa de contenedor para visualizar los vídeos.

Incluye, además, los típicos controladores de reproducción, como play, pausar, avanzar o retroceder en la reproducción, etc. Y además, se han añadido unas opciones extra como un botón de “me gusta” o “no me gusta” y selector de calidad.

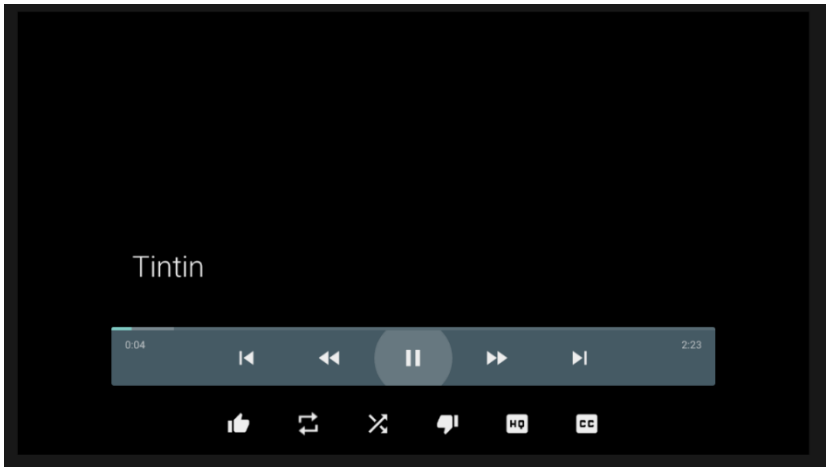


*Ilustración 44. Pantalla de reproductor.*

Es importante destacar que, en el emulador, al principio no fue posible reproducir ningún vídeo de alguna fuente remota. El reproductor actúa correctamente, ya que recibe la URI de los vídeos y obtiene la duración del mismo, como se aprecia en la

imagen. No se visualiza el vídeo, pero se escucha el audio, aunque no del todo correctamente.

Sin embargo, tras investigar este hecho se llegó a la conclusión de que el problema es debido alto consumo de recursos, por lo que se probó con un vídeo de muy baja calidad y se verificó que se reproducía correctamente.

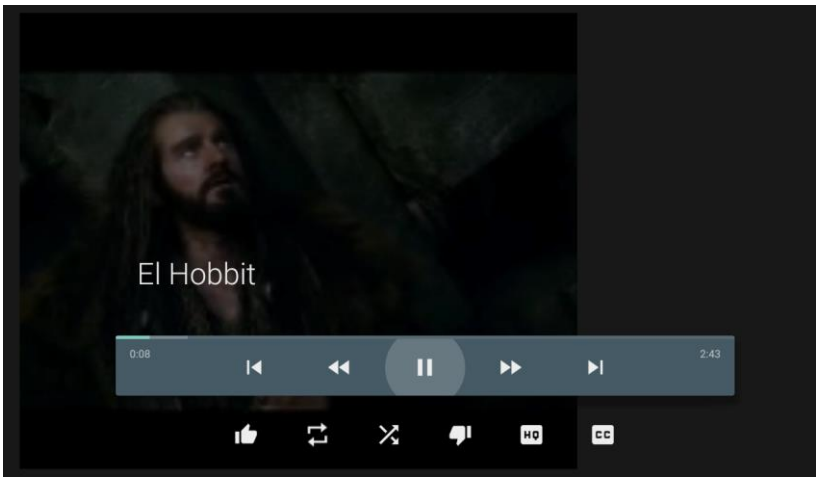


*Ilustración 45. Reproducción correcta en emulador.*

En el capítulo 5, se tratará el tema de las diferentes limitaciones del emulador y se comentará el problema de la reproducción en streaming.

### Pantalla de búsqueda

En este apartado se va a explicar todo lo referente a la pantalla de búsqueda de la aplicación. Como es de esperar, en esta pantalla se pueden realizar búsquedas, en este caso por el título de la película, y se muestran los resultados obtenidos, en caso de que existan o un mensaje indicando que no se han encontrado resultados.

Para estos casos, es muy recomendable utilizar el *fragment* SearchFragment de la librería Leanback v17, pensado específicamente para este propósito.

```
public class SearchTVFragment extends SearchFragment implements
SearchFragment.SearchResultProvider {…}
```

Este *fragment* define un cuadro de búsqueda para introducir el texto y un icono que, al pulsarlo, permite la entrada de texto por voz. Esta última característica no está disponible en el emulador.

Primero, se guarda en una lista, todas las películas definidas en la aplicación para realizar la búsqueda sobre dicha lista, utilizando el mismo método que en el apartado de la vista de rejilla.

Para realizar la búsqueda se definen dos métodos que obtienen el texto introducido por el usuario en el cuadro de búsqueda:

El primero, **onQueryTextChange():**

```
public boolean onQueryTextChange(String newQuery) {
        mRowsAdapter.clear();
        if (!TextUtils.isEmpty(newQuery)) {
            mDelayedLoad.setSearchQuery(newQuery);
            mHandler.removeCallbacks(mDelayedLoad);
            mHandler.postDelayed(mDelayedLoad, SEARCH_DELAY_MS);
        }
        return true;
    }
```

Y **onQueryTextSubmit()**:

```
public boolean onQueryTextSubmit(String query) {
        mRowsAdapter.clear();
        if (!TextUtils.isEmpty(query)) {
            mDelayedLoad.setSearchQuery(query);
            mHandler.removeCallbacks(mDelayedLoad);
            mHandler.postDelayed(mDelayedLoad, SEARCH_DELAY_MS);
        }
        return true;
    }
```

El primero obtiene el texto introducido a medida que el usuario está escribiendo, con un tiempo de espera definido en la constante SEARCH_DELAY_MS. El segundo, obtiene el texto una vez que el usuario pulsa el botón de buscar.

Ambos métodos obtienen el texto y ejecutan la siguiente instrucción:

```
mDelayedLoad.setSearchQuery(query);
```

Donde mDelayedLoad se trata de un objeto *Runnable* de Android. Este tipo de objeto está pensado para que todos sus métodos se lleven a cabo en un hilo independiente.

Por ello, cuando se ejecuta esta instrucción, en otro hilo se realiza la búsqueda con el texto introducido, almacenado en la variable *query.*

```
public void run() {
  mRowsAdapter.clear();
  ArrayObjectAdapter adapter = new ArrayObjectAdapter(new CardPresenter());
  List<Movie> movies = loadMovies();
  String title = "";
  Movie found = null;
    for(int i = 0; i<movies.size(); i++){
                title = movies.get(i).getTitle();
                if(title.equals(query)){
                    found = movies.get(i);
                    adapter.add(0, found);
                    HeaderItem header = new HeaderItem(0,
getResources().getString(R.string.search_results), null);
                    mRowsAdapter.add(new ListRow(header, adapter));
                }
            }
            if (found == null) Toast.makeText(getActivity(), "No hay
resultados", Toast.LENGTH_LONG).show();
             }
```

Se recorre la lista completa de películas y se obtiene el título de cada una de ellas, en caso de coincidir con el texto de la variable *query,* mediante un *adapter* se define el HeaderItem, igual que las categorías del catálogo, pero en este caso sólo habrá uno, con el texto "Resultados de la búsqueda" y mediante el *presenter* de tipo tarjeta, *CardPresenter,* se visualizan los resultados.

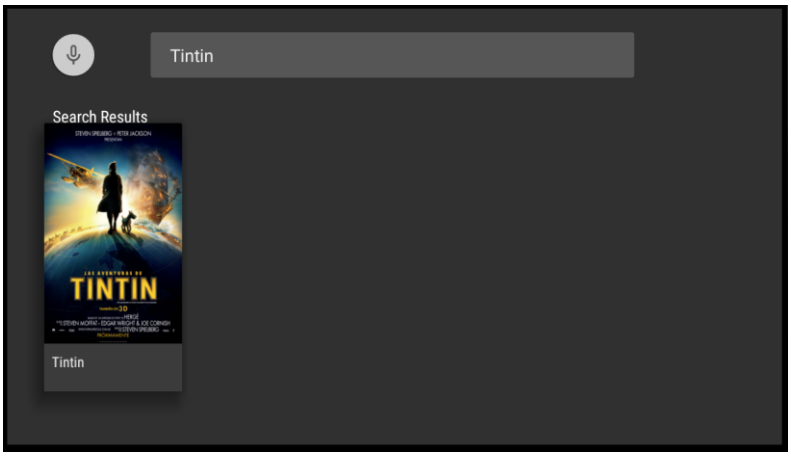


*Ilustración 46. Búsqueda realizada con éxito.*

En caso de no encontrar ningún resultado se mostrará un mensaje indicándolo, mediante el uso del elemento Android llamado *Toast,* pensado precisamente para mostrar pequeños mensajes al usuario.

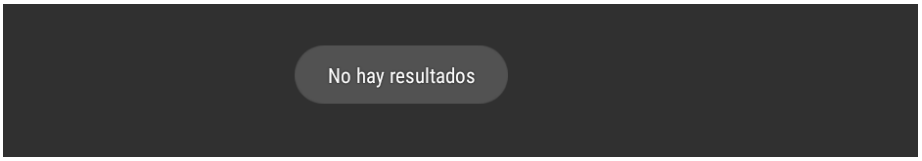


*Ilustración 47. Resultado de búsqueda no satisfactoria.*

Por último, cuando la búsqueda tiene resultados, se puede acceder a ellos y se visualizará la *activity* de detalles, con la información de la película seleccionada.

# Capítulo 5: Plan de Pruebas.

En este capítulo se describirán las pruebas realizadas para garantizar el correcto funcionamiento de la aplicación. Tras el diseño e implementación, se llevarán a cabo una serie de pruebas para comprobar que la aplicación funciona tal y como se espera.

Todas las pruebas se han realizado en el emulador de Android, de Android Studio, ya que actualmente no hay ningún dispositivo a la venta con tecnología Android TV.

Para ello se han definido unas sencillas tablas explicativas para facilitar la lectura. Primero se ha comprobado el correcto funcionamiento de cada *activity* por separado, mediante pruebas unitarias. Por último se ha realizado una serie de pruebas de integración, para verificar que todas las *activities* interactúan entre ellas de forma correcta.

## *5.1 Pruebas unitarias*

**Activity principal (catálogo)**

| Identificador | Descripción | Resultado |
|---|---|---|
| 1 | Al ejecutar la aplicación, se carga la *activity* principal y se visualiza el catálogo correctamente. | Superada |
| 2 | Se visualizan las diferentes categorías definidas y sus películas correspondientes, así como los ajustes, cada uno mostrado con su *presenter* correspondiente. | Superada |
| 3 | Se comprueba que la navegación es correcta por las diferentes categorías y elementos del catálogo. | Superada |
| 4 | Dentro de una categoría se verifica que se puede navegar por las demás categorías, sin necesidad de regresar y seleccionar la nueva categoría. | Superada |
| 5 | Durante la navegación, se comprueba que los elementos aparecen resaltados para que el usuario tenga constancia en todo momento del elemento en el que está posicionado. | Superada |
| 6 | Los datos de cada película se cargan correctamente: portada y título, y se comprueba que se visualizan correctamente con el formato del *presenter* asociado. | Superada |
| 7 | Se comprueba que los elementos del apartado de opciones se visualizan correctamente. | Superada |
| 8 | Mediante las flechas de dirección se comprueba que es sencillo alcanzar el icono de búsqueda. | Superada |

*Tabla 4. Pruebas unitarias de la activity principal.*

***Activity* vista de detalles**

| Identificador | Descripción | Resultado |
|---|---|---|
| 1 | Al lanzar la *activity* se comprueba que se visualice la información correspondiente a la película seleccionada. | Superada |
| 2 | La información de la película: portada, título y sinopsis se visualiza con el formato correcto impuesto por el *presenter* correspondiente. | Superada |
| 3 | Se comprueba que la imagen de fondo de la aplicación cambia y se visualiza la imagen asociada a la película seleccionada. | Superada |
| 4 | Se comprueba que se visualice el botón que da acceso a la visualización del tráiler y que es sencillo llegar hasta él mediante las flechas de dirección. | Superada |
| 5 | Se comprueba que navegando hacia abajo con las flechas de dirección, bajo la sección que muestra la información de la película, aparece el *fragment* que contiene el listado de películas relacionadas. | Superada |
| 6 | En el listado de películas relacionadas, se comprueba que se puede navegar por los elementos correctamente. | Superada |

*Tabla 5. Pruebas unitarias de la activity de vista de detalles.*

***Activity* vista de rejilla.**

| Identificador | Descripción | Resultado |
|---|---|---|
| 1 | Se accede y se comprueba que aparecen todas las películas disponibles. | Superada |
| 2 | Las películas se visualizan con el formato del *presenter* correspondiente. | Superada |
| 3 | Se navega por todos los elementos, mediante las flechas de dirección y se comprueba que resalta el elemento donde el usuario esté posicionado. | Superada |
| 4 | Cada uno de los elementos debe mostrar su portada y título correspondiente. | Superada |
| 5 | Al posicionarse sobre cualquier elemento, se cambia la imagen de fondo y se comprueba que se visualice la imagen referente al elemento seleccionado. | Superada |

*Tabla 6. Pruebas unitarias de la activity de vista de rejilla.*

***Activity* buscador.**

| Identificador | Descripción | Resultado |
|---|---|---|
| 1 | Se comprueba que se carga el cuadro de búsqueda para introducir el texto, con el formato definido por el tipo de *fragment*. | Superada |
| 2 | Para realizar una búsqueda se verifica que se puede escribir en dicho cuadro de texto. | Superada |
| 3 | Se introduce texto en la búsqueda y se comprueba que, en el momento, en que coincida con el título de alguna película, muestra los resultados. | Superada |
| 4 | Se introduce un texto a buscar y se pulsa *enter*, se comprueba que si no hay resultados, aparece un mensaje indicándolo al usuario. | Superada |
| 5 | Una vez realizada una búsqueda con resultados, éstos aparecen en la parte inferior con el formato definido por el *presenter,* carátula y título. | Superada |
| 6 | Se comprueba que se puede navegar por los diferentes resultados obtenidos en una búsqueda exitosa. | Superada |

*Tabla 7. Pruebas unitarias de la activity del buscador de contenido.*

***Activity* del reproductor.**

| Identificador | Descripción | Resultado |
|---|---|---|
| 1 | Al acceder a la visualización de un tráiler, se carga el reproductor con todos los controles para la reproducción. | Superada |
| 2 | Se pulsa el botón de *play* y el reproductor obtiene el vídeo y carga su duración. | Superada |
| 3 | El vídeo se visualiza. | Superada |

*Tabla 8. Pruebas unitarias de la activity del buscador de contenido.*

## *5.2 Pruebas de integración entre activities*

| Identificador | Descripción | Resultado |
|---|---|---|
| 1 | Desde la *activity* principal, se accede a alguna de las películas y se comprueba que se lance la *activity* de detalles con el contenido de la película seleccionada. | Superada |
| 2 | Desde la *activity* principal se navega hasta el icono de búsqueda y se accede. Se comprueba que se lanza la *activity* de búsqueda. | Superada |
| 3 | Desde la *activity* principal se accede a la opción de vista de rejilla t se comprueba que se lanza su *activity* asociada. | Superada |
| 4 | Estando en la *activity* de detalles, pulsando botón *atrás*, se comprueba que la aplicación regresa a la *activity* principal. | Superada |
| 5 | Desde la *activity* de detalles, se selecciona la opción de ver tráiler y se comprueba que se lanza la *activity* del reproductor. | Superada |
| 6 | Desde la *activity* de detalles, se accede a algún elemento de la lista de contenido relacionado y se comprueba que se recarga la propia *activity* pero mostrando los detalles del nuevo elemento seleccionado. | Superada |
| 7 | Desde la *activity* del reproductor, se pulsa *atrás* y se comprueba que se lanza la *activity* de detalles. | Superada |
| 8 | Desde la *activity* de búsqueda se pulsa el botón *atrás,* y se comprueba que se lanza la *activity* principal. | Superada |
| 9 | Desde la *activity* de búsqueda, tras realizar una búsqueda, se accede a alguno de los resultados y se comprueba que se lanza la *activity* de detalles con el contenido de la película seleccionada. | Superada |

*Tabla 9. Pruebas de integración entre las diferentes activities.*

## *5.3 Resultados y limitaciones.*

A la vista de los resultados de las diferentes pruebas realizadas en el emulador de Android, el resultado global es satisfactorio.

Toda la interfaz de usuario de la aplicación mantiene un diseño adecuado para un televisor en todo momento. La navegación ha resultado ser muy sencilla e intuitiva para el usuario, y en ningún momento se ha tenido la sensación de no saber cómo llegar a ningún elemento en pantalla.

La funcionalidad, exceptuando la reproducción de vídeos, ha sido satisfactoria, y se han alcanzado los objetivos y cumplido los requisitos propuestos, y más teniendo en cuenta lo que implica el desarrollo para una tecnología tan sumamente nueva.

En cuanto a limitaciones, cabe destacar de nuevo que la aplicación se ha desarrollado y testeado gracias al emulador de Android incluido en el entorno Android Studio, con todo lo que ello conlleva. Los emuladores, por norma general, trabajan bastante bien y cumplen sobradamente para el testeo de la mayoría de aplicaciones, pero siempre es muy recomendable el uso de dispositivos físicos, ya que la emulación de cualquier entorno requiere muchos recursos a nivel hardware y no es igual de estable que un entorno físico.

En este caso concreto, no ha sido imposible realizar el testeo en un televisor Android TV, ya que de momento, no ha comenzado su comercialización.

El hecho de testear la aplicación en el emulador ha dejado notar ciertas limitaciones:

- Requiere un buen ordenador, con buen hardware. La aplicación se ha testeado en un PC con procesador i7 y 4GB de memoria RAM y, aun así, el inicio del emulador es bastante lento y el uso de memoria RAM nunca bajaba del 80% aproximadamente.
- Se puede apreciar que la fluidez de la aplicación se ve mermada.
- Lo más importante de todo, la reproducción de vídeo desde un recurso en línea, como es el caso de los tráilers de las películas, ha sido bastante tediosa. Al principio se pensó que era debido a algún error en el desarrollo, pero se comprobó que todo estaba correcto. A raíz de ello, se investigó sobre ese tema y todas las soluciones apuntaban a que era necesario su testeo en un dispositivo físico [44] [45]. Finalmente, como se comentó anteriormente, se consiguió visualizar vídeos con muy baja calidad, en formato 3GP y se verificó que el hecho de no poder reproducir vídeos de mayor calidad era debido al alto consumo de recursos y las limitaciones que tiene su testeo en un emulador.

Dejando de lado los detalles mencionados anteriormente, el emulador funciona correctamente y verifica sin problemas el correcto funcionamiento de la aplicación.

Como se ha explicado anteriormente, no hay posibilidad de su testeo en un Android TV real. Sin embargo, se pensó que se podría testear en un dispositivo con Android 5.0, ya que es el nivel de API mínimo que requiere Android TV.

En este caso, tampoco se disponía de un terminal con dicha versión de Android, pero se pudo conseguir uno prestado, el smartphone de Google, Nexus 4, con el que se realizó el testeo. Para poder ejecutar una aplicación pensada para TV en un móvil, hubo que retocar ciertos aspectos de la interfaz. Aunque su visualización no era óptima, permitió navegar hasta la *activity* con el reproductor y se pudo comprobar que los vídeos se reproducen correctamente, tal y como se pensaba. Con ello, quedó verificado que el problema es debido a las limitaciones del emulador a la hora de testear ciertas funcionalidades que requieren un elevado uso de recursos hardware.

Todos estos problemas a la hora de desarrollar para Android TV desaparecerán cuando comience la comercialización de televisores con dicha tecnología y todas las pruebas podrán realizarse directamente en dichos televisores.

# Capítulo 6: Conclusiones y trabajos futuros.

Después del fracaso que supuso Google TV, parece que por fin, la multinacional estadounidense ha apostado fuerte por su expansión a dispositivo como los televisores, gracias a Android TV. Esta nueva plataforma promete ser una de las grandes alternativas en el ámbito de las Smart TV.

En cuanto al desarrollo, Google proporciona una documentación muy completa y las librerías específicas para Android TV, facilitan en gran medida el diseño y desarrollo. Gracias al uso de los diferentes *fragments* y *presenters* predefinidos, se pueden construir aplicaciones totalmente orientadas para la televisión con poco esfuerzo, al mismo tiempo que se pueden definir *fragments* y *presenters* personalizados.

La gran expansión del sistema operativo Android también es un punto a favor, ya que proporciona el acceso a una gran colección de aplicaciones y, al mismo tiempo, facilita en gran medida el desarrollo para esta plataforma. Para los desarrolladores Android será muy sencillo construir aplicaciones para esta plataforma e, incluso, adaptar aplicaciones existentes.

En el caso de la aplicación desarrollada en este proyecto, actualmente, el contenido está definido localmente, aunque se obtengan recursos de internet. Como mejoras o trabajos futuros, cabe destacar, por encima de todo, que el siguiente paso sería hacer la aplicación dinámica. Como propuesta, la aplicación podría conectar con una base de datos donde se almacena toda la información de las películas. Por cada una de las categorías, se realizaría una consulta a dicha base de datos. El contenido devuelto, se puede almacenar en diferentes formatos, como podría ser un archivo XML o JSON. A partir de la información contenida en ese fichero, se cargarían los elementos del catálogo. Del mismo modo, con cualquier actualización en la base de datos, el contenido obtenido por la aplicación quedaría actualizado también. También en esta línea, la búsqueda de contenido se debería realizar mediante consultas directamente a la base datos y que la búsqueda no se realizase dentro de la aplicación.

Por otra parte, se puede extender el uso de la aplicación, añadiendo funcionalidades como el alquiler o compra de películas.

## Entorno socio-económico

En esta sección se van a comentar algunos de los aspectos y datos más relevantes sobre la expansión de los diferentes sistemas operativos móviles y, en especial, de Android, que es la base de la tecnología aquí estudiada, Android TV.

El primer *smartphone* con este sistema operativo apareció en el año 2008 y, desde entonces, el crecimiento de dicho sistema ha aumentado hasta alcanzar una cuota de mercado de más del 83%.

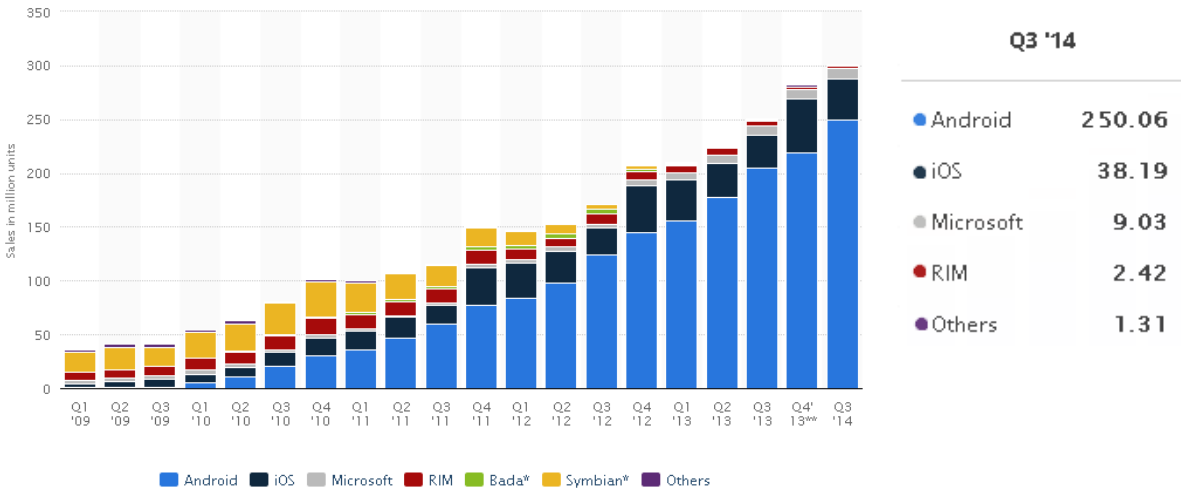


*Ilustración 48. Ventas de smartphones globales. Fuente: statista.com*

Tal y como se aprecia en la ilustración, la tendencia de crecimiento de Android ha aumentado con el paso de los años. A partir del año 2011, Android se posicionó como el sistema con mayor cuota de mercado en *smartphones* y, desde entonces, esa cuota no ha hecho más que aumentar. Los últimos datos, de tercer cuarto del año 2014, indican que se vendieron alrededor de 300 millones de *smartphones* a nivel global, de los cuales, unos 250 millones utilizaban Android como sistema operativo, una cuota de más del 83%. Mientras, la tendencia de su principal competidor actualmente, iOS, el sistema de la compañía Apple, ha sido bastante constante y siempre mantiene una cuota entre 13% y 18% [46].

Por otra parte, la venta de Smart TV también ha aumentado considerablemente con el paso de los años. En la siguiente ilustración, se aprecia dicho aumento desde el año 2011 y un pronóstico para los próximos años.

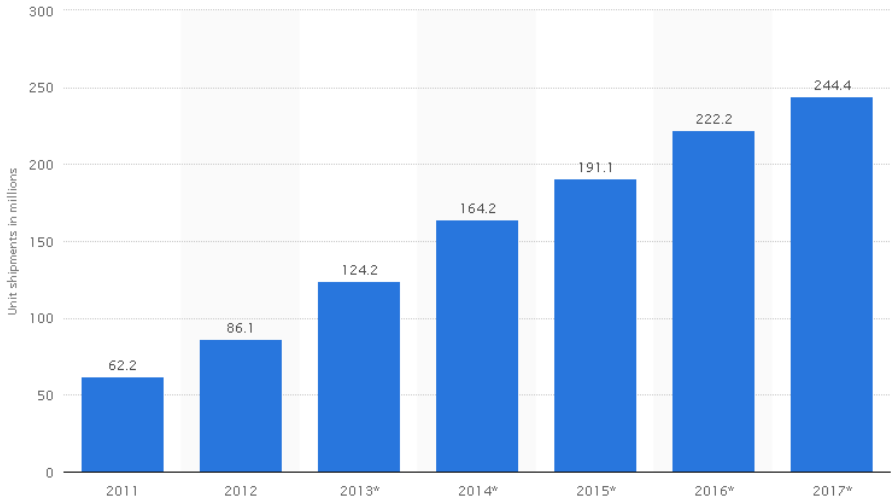


*Ilustración 49. Pronóstico de ventas Smart TV. Fuente: statista.com*

De cumplirse los pronósticos, en el año 2017, la venta de Smart TV habrá aumentado aproximadamente un 400% respecto a 2011 [47].

A la vista de ambos estudios, queda patente un claro crecimiento tanto la comercialización de Smart TV como en la expansión del sistema operativo Android. Por ello, la combinación de ambos términos parece una muy buena alternativa.

## Marco regulador


En el año 1989 se definió una directiva denominada *Televisión sin fronteras (TSF)*, que se trata del pilar básico que asienta la regulación audiovisual europea. Se basa en dos principios básicos: la libre circulación de programas televisivos europeos en el mercado interior y la obligación de las cadenas de televisión de dedicar, siempre que sea posible, más de la mitad de su tiempo de emisión a obras europeas. La Directiva pretende garantizar la libre circulación de los servicios de radiodifusión en el mercado interior preservando algunos objetivos importantes de interés público, como la diversidad cultural, el derecho de réplica, la protección de los consumidores y la protección de los menores [48].

Con la llegada de los nuevos contenidos multimedia vía internet, dicha directiva fue modificada en el año 2007 para adaptarla a los nuevos tiempos y pasó a denominarse directiva de *Servicios de medios audiovisuales sin fronteras.* En esta nueva directiva europea ya se hizo la distinción entre:

- Servicios lineales: que designan los servicios de televisión tradicional, Internet, y telefonía móvil que los telespectadores reciben pasivamente.
- Servicios no lineales: servicios de televisión a la carta o bajo demanda (VOD) que los telespectadores pueden escoger.

En esta directiva se simplificó y modernizó el marco regulatorio de los servicios lineales y se establecieron unas normas mínimas para los servicios no lineales, sobre todo, en materia de protección de menores, prevención de actitudes racistas y prohibición de publicidad encubierta [49].

*"Esta nueva regulación comunitaria es, en la práctica, la adecuación del marco regulador al escenario convergente que resulta de la incorporación de las nuevas tecnologías a la transmisión y difusión de servicios de comunicación audiovisual en la era digital."* [50]

# Presupuesto

A continuación se presenta una estimación detallada del presupuesto necesario para el desarrollo de este trabajo.

Los costes se presentan en tres apartados, licencias software, equipos y personal. No se incluyen costes de dietas o viajes y el tiempo de amortización estimado para los equipos será de 5 años y 3 en el caso de las licencias. La tasa de costes indirectos será del 20% y un IVA del 21%. El coste de personal incluye los costes para la empresa como el alta en la seguridad social.

| DESCRIPCIÓN | COSTE | DEDICACIÓN | COSTE IMPUTABLE |
|---|---|---|---|
| **ADOBE PHOTOSHOP CC** | 12€/mes | 5 meses | 60,00€ |
| **OFFICE HOGAR Y EMPRESAS 2013** | 269€ | 5 meses / 3 años = 14% | 37,66€ |
| **VISIO PROFESSIONAL 2013** | 739€ | 5 meses / 3 años = 14% | 103,46€ |
| **PROJECT STANDARD 2013** | 769€ | 5 meses / 3 años = 14% | 107,66€ |
| **ENTERPRISE ARCHITECT 11** | 210€ | 5 meses / 3 años = 14% | 29,4€ |
| **TOTAL** | | | **338,18€** |

*Tabla 10. Costes por licencia software.*

| DESCRIPCIÓN | COSTE | DEDICACIÓN | COSTE IMPUTABLE |
|---|---|---|---|
| **PC DE SOBREMESA** | 1000€ | 5 meses / 5 años = 8,3% | 83,33€ |
| **TOTAL** | | | **83,33€** |

*Tabla 11. Costes por equipos hardware.*

| DESCRIPCIÓN | COSTE | DEDICACIÓN | COSTE IMPUTABLE |
|---|---|---|---|
| **INGENIERO SENIOR** | 100€/hora | 20h/mes durante 5 meses | 10.000€ |
| **INGENIERO JUNIOR** | 50€/hora | 100h/mes durante 5 meses | 25.000€ |
| **TOTAL** | | | **35.000€** |

*Tabla 12. Costes por personal.*

| DESCRIPCIÓN | COSTE IMPUTABLE | IVA (21%) | COSTE FINAL |
|---|---|---|---|
| **SOFTWARE** | 338,18€ | 71€ | 409,18€ |
| **EQUIPOS** | 83,33€ | 17,5€ | 100,83€ |
| **PERSONAL** | 35.000,00€ | 7350,00€ | 42.350,00€ |
| **COSTES INDIRECTOS (20%)** | 7084,30€ | 1487,70€ | 8.572,00€ |
| **TOTAL** | | | **51.432,01€** |

*Tabla 13. Costes totales.*

# Lista de tablas



# Lista de ilustraciones

# Bibliografía

[1] D. T. d. C. ". Acacias", «tiscar.wikispaces.com,» 2008. [En línea]. Available: https://tiscar.wikispaces.com/file/view/4.1television.pdf. [Último acceso: 2 Febrero 2015].

[2] e. y. t. d. E. Ministerio de industria, «¿Qué es la TDT?,» [En línea]. Available: http://www.televisiondigital.gob.es/TDT/Paginas/que-es-tdt.aspx. [Último acceso: 15 Diciembre 2014].

[3] M. Rouse, «video on demand (VoD),» [En línea]. Available: http://searchtelecom.techtarget.com/definition/video-on-demand. [Último acceso: 18 Diciembre 2014].

[4] T. Pendlebury, «Smart TV: what you need to know,» Julio 2011. [En línea]. Available: http://www.cnet.com/au/news/smart-tv-what-you-need-to-know/. [Último acceso: 3 Enero 2015].

[5] J. Evans, «What is Smart TV, and why Does it Matter?,» 2011. [En línea]. Available: http://www.huffingtonpost.co.uk/jonny-evans/what-is-smart-tv-and-why-_b_951716.htmll. [Último acceso: 3 Enero 2015].

[6] C. Janssen, «Internet Protocol Television (IPTV),» [En línea]. Available: http://www.techopedia.com/definition/24957/internet-protocol-television-iptv. [Último acceso: 3 Enero 2015].

[7] DLNA.ORG, «DLNA Official Website,» [En línea]. Available: http://www.dlna.org/. [Último acceso: 3 Enero 2015].

[8] M. Flacy, «BEST SMART TV PLATFORMS FOR CORD CUTTERS,» 2013. [En línea]. Available: http://www.digitaltrends.com/home-theater/best-smart-tv-platforms-for-cord-cutters/. [Último acceso: 4 Enero 2015].

[9] J. ZUÑIGA, «A la espera del gran desarrollo de APP's para Smart TV,» 2014. [En línea]. Available: http://www.appsmag.es/1896/a-la-espera-del-gran-desarrollo-de-apps-para-smart-tv/. [Último acceso: 4 Enero 2015].

[10] Informatica-Hoy, «Conoces lo qué es un Smart TV?,» 2013. [En línea]. Available: http://www.informatica-hoy.com.ar/electronica-consumo-masivo/Que-es-Smart-TV.php. [Último acceso: 4 Enero 2015].

[11] S. Barton, «Samsung Smart TV 2014 – All the key new features detailed,» 2014. [En línea]. Available: http://www.expertreviews.co.uk/tvs-entertainment/tvs/27479/samsung-smart-tv-2014-all-the-key-new-features-detailed. [Último acceso: 4 Enero 2015].

[12] Samsung, «Samsung Smart TV,» [En línea]. Available: http://www.samsung.com/us/experience/smart-tv/. [Último acceso: 4 Enero 2015].

[13] LG, «NetCast Entertainment Access,» [En línea]. Available: http://www.lg.com/us/netcast/index.jsp. [Último acceso: 4 Enero 2015].

[14] LG, «LG PRESENTA NETCAST™, EL NUEVO ACCESO ONLINE A CONTENIDOS DESDE LA TV,» 2011. [En línea]. Available: http://mylg.com.ar/blog/2010/11/lg-presenta-netcast%E2%84%A2-el-nuevo-acceso-

online-a-contenidos-desde-la-tv/. [Último acceso: 4 Enero 2015].

[15] Xataka, «Google TV,» [En línea]. Available: http://www.xataka.com/gadgets/accesorios/google-tv. [Último acceso: 4 Enero 2015].

[16] J. Diaz, «Android Headliner: What Happened To Google TV?,» 11 Mayo 2014. [En línea]. Available: http://androidheadlines.com/2014/05/ah-prime-time-happened-google-tv.html. [Último acceso: 4 Enero 2015].

[17] K. Puerto, «Las teles Samsung se pasan a Tizen en 2015,» 1 Enero 2015. [En línea]. Available: http://www.xataka.com/televisores/las-teles-samsung-se-pasan-a-tizen-en-2015. [Último acceso: 4 Enero 2015].

[18] Samsung, «Samsung Electronics Redefines TV Experience with New Smart TV Powered by Tizen,» 1 enero 2015. [En línea]. Available: http://global.samsungtomorrow.com/samsung-electronics-redefines-tv-experience-with-new-smart-tv-powered-by-tizen/. [Último acceso: 5 Enero 2015].

[19] T. org, «Tizen,» 2012. [En línea]. Available: https://www.tizen.org/es/about/devices. [Último acceso: 5 Enero 2015].

[20] LG, «LG WebOS,» [En línea]. Available: http://www.lg.com/es/webos. [Último acceso: 5 Enero 2015].

[21] Mozilla, «Firefox OS Unlocks the Power of the Web as the Platform Across Expanding Ecosystem of Partners and Devices,» 5 Enero 2015. [En línea]. Available: https://blog.mozilla.org/blog/2015/01/05/firefoxosces2015/. [Último acceso: 8 Enero 2015].

[22] Wikipedia, «Set-Top-Box,» [En línea]. Available: http://es.wikipedia.org/wiki/Set-top_box. [Último acceso: 8 Enero 2015].

[23] Apple, «Apple TV,» [En línea]. Available: https://www.apple.com/es/appletv/. [Último acceso: 8 Enero 2015].

[24] A. Adictos, «Análisis – Nuevo Apple TV: el complemento perfecto,» [En línea]. Available: http://www.appleadictos.com/especiales/especial-analisis-nuevo-apple-tv-el-complemento-perfecto/. [Último acceso: 8 Enero 2015].

[25] Amazon, «Amazon Fire TV,» [En línea]. Available: http://www.amazon.com/Fire-TV-streaming-media-player/dp/B00CX5P8FC. [Último acceso: 8 Enero 2015].

[26] Roku, «Roku 3,» [En línea]. Available: https://www.roku.com/products/roku-3. [Último acceso: 8 Enero 2015].

[27] Google, «Google Chromecast,» [En línea]. Available: https://www.google.es/chrome/devices/chromecast/. [Último acceso: 8 Enero 2015].

[28] G. Inc., «Android TV,» 2014. [En línea]. Available: http://www.android.com/tv/. [Último acceso: 9 Enero 2015].

[29] J. Penalva, «Android TV está listo para buscar el éxito que no tuvo Google TV,» 6 Abril 2014. [En línea]. Available: http://www.xataka.com/televisores/android-tv-esta-listo-para-buscar-el-exito-que-no-tuvo-google-tv. [Último acceso: 9 Enero 2015].

[30] FOliveros, «Cara a cara: Nexus Player vs Apple TV,» 15 Octubre 2014. [En línea]. Available: http://hipertextual.com/archivo/2014/10/nexus-player-vs-apple-tv/. [Último acceso: 9 Enero 2015].

[31] Wikipedia, «List of smart TV platforms and middleware software,» 2015. [En línea]. Available: http://en.wikipedia.org/wiki/List_of_smart_TV_platforms_and_middleware_software. [Último acceso: 9 Enero 2015].

[32] A. 4. A. D. Essentials, «An Overview of the Android Architecture,» [En línea]. Available: http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture. [Último acceso: 10 Enero 2015].

[33] A. A. C. Tutorialspoint, «Android Application Components,» [En línea]. Available: http://www.tutorialspoint.com/android/android_application_components.htm. [Último acceso: 10 Enero 2015].

[34] G. Inc., «Activities,» [En línea]. Available: http://developer.android.com/intl/es/guide/components/activities.html. [Último acceso: 10 Enero 2015].

[35] G. Inc., «Services,» [En línea]. Available: http://developer.android.com/intl/es/guide/components/services.html. [Último acceso: 10 Enero 2015].

[36] G. Inc., «Broadcast Receivers,» [En línea]. Available: http://developer.android.com/intl/es/reference/android/content/BroadcastReceiver.html. [Último acceso: 10 Enero 2015].

[37] G. Inc., «Content Providers,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/providers/content-providers.html. [Último acceso: 10 Enero 2015].

[38] G. Inc., «Intents and Intent Filters,» [En línea]. Available: http://developer.android.com/guide/components/intents-filters.html. [Último acceso: 10 Enero 2015].

[39] X. developer, «Working with AndroidManifest.xml,» [En línea]. Available: http://developer.xamarin.com/guides/android/advanced_topics/working_with_androidmanifest.xml/. [Último acceso: 10 Enero 2015].

[40] G. Inc., «View,» [En línea]. Available: http://developer.android.com/intl/es/reference/android/view/View.html. [Último acceso: 10 Enero 2015].

[41] G. Inc., «Layouts,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/ui/declaring-layout.html. [Último acceso: 10 Enero 2015].

[42] G. Inc., «Fragments,» [En línea]. Available: http://developer.android.com/intl/es/guide/components/fragments.html. [Último acceso: 15 Enero 2015].

[43] G. Inc., «App Manifest,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/manifest/uses-sdk-element.html. [Último acceso: 15 Enero 2015].

[44] J. -. Stackoverflow, «How to Play mp4 Video in Android Emulator Using Remote URL?,» [En línea]. Available: http://stackoverflow.com/questions/6582016/how-to-play-mp4-video-in-android-emulator-using-remote-url. [Último acceso: 20 Enero 2015].

[45] C. -. Stackoverflow, «How to display Video in the Android Emulator from Remote URL?,» [En línea]. Available: http://stackoverflow.com/questions/1425502/how-to-display-video-in-the-android-emulator-from-remote-url. [Último acceso: 20 Enero 2015].

[46] Statista, «Global smartphone sales to end users from 1st quarter 2009 to 3rd quarter 2014, by operating system (in million units),» 2015. [En línea]. Available: http://www.statista.com/statistics/266219/global-smartphone-sales-since-1st-quarter-2009-by-operating-system/. [Último acceso: 10 Febrero 2015].

[47] Statista, «Forecast of worldwide smart TV unit shipments from 2011 to 2017 (in millions),» 2015. [En línea]. Available: http://www.statista.com/statistics/314616/smart-tv-unit-shipment-worldwide-forecast/. [Último acceso: 15 Febrero 2015].

[48] S. d. l. l. d. l. UE, «Actividades de radiodifusión televisiva: Directiva Televisión sin fronteras (TSF),» 2008. [En línea]. Available: http://europa.eu/legislation_summaries/audiovisual_and_media/l24101_es.htm. [Último acceso: 19 Febrero 2015].

[49] S. d. l. l. d. l. UE, «Directiva Servicios de medios audiovisuales sin fronteras,» 2008. [En línea]. Available: http://europa.eu/legislation_summaries/audiovisual_and_media/l24101a_es.htm. [Último acceso: 19 Febrero 2015].

[50] Á. G. Castillejo, «Revista TELOS - La regulación de los contenidos audiovisuales,» 2010. [En línea]. Available: http://telos.fundaciontelefonica.com/url-direct/pdf-generator?tipoContenido=articuloTelos&idContenido=2010110310590001&idioma=es. [Último acceso: 19 Febrero 2015].

# Anexo 1.

## Creación de un proyecto para Android TV mediante Android Studio.

En este anexo se va a incluir una guía de configuración del entorno Android Studio para la creación de un proyecto para el desarrollo en Android TV, incluyendo una breve explicación para la configuración de un emulador para el testeo de aplicaciones de dicha tecnología. Cabe destacar que para la creación de esta guía se ha utilizado la versión 1.01 del entorno Android Studio.

Antes de nada, es necesario configurar el SDK de Android que utilizará el entorno, ya que para el desarrollo en Android TV es necesario, como mínimo, la versión 24.0.0 del SDK tolos y el API 21 de Android 5.0 *Lollipop.* En la ilustración 50 se muestran los paquetes instalados del *SDK Manager*.

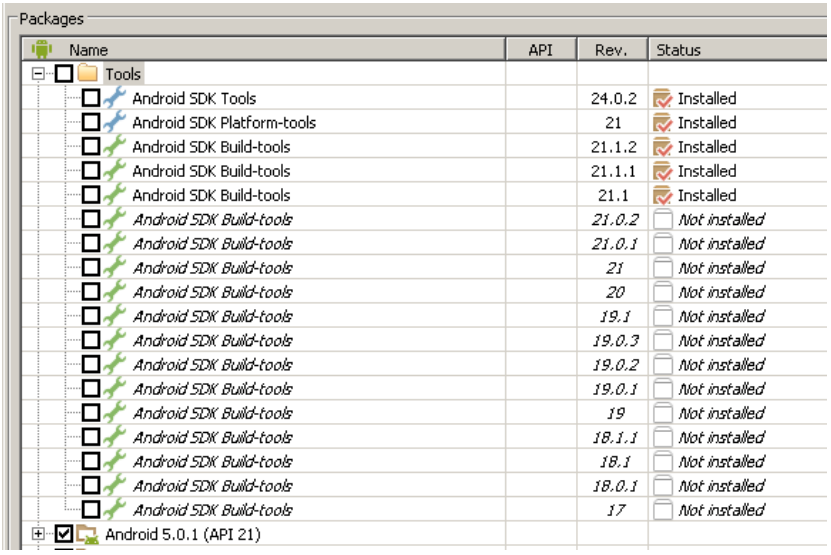


*Ilustración 49. SDK Manager, SDK Tools 24.0.2 y API 21*

Una vez descargados los paquetes con el SDK Manager ya se puede configurar el entorno Android Studio para generar un proyecto para Android TV. En la ilustración 51 se muestra la vista principal, para la creación de nuevos proyectos, abrir proyectos existentes o importarlos y algunas opciones de configuración.

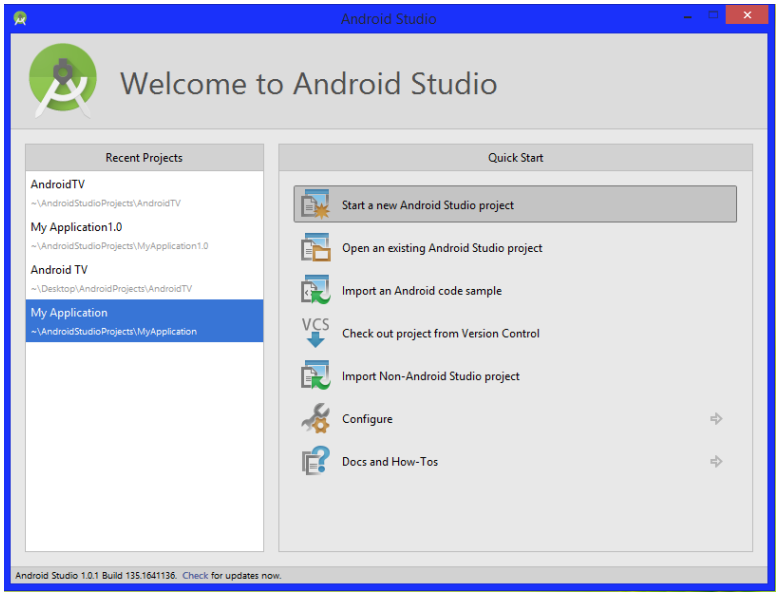


*Ilustración 50. Vista principal.*

En este caso se selecciona la primera opción para crear un proyecto nuevo y en la siguiente pantalla se indica tanto el nombre del proyecto como del paquete y la ruta donde se almacenará. La ilustración 52, muestra dicha pantalla.

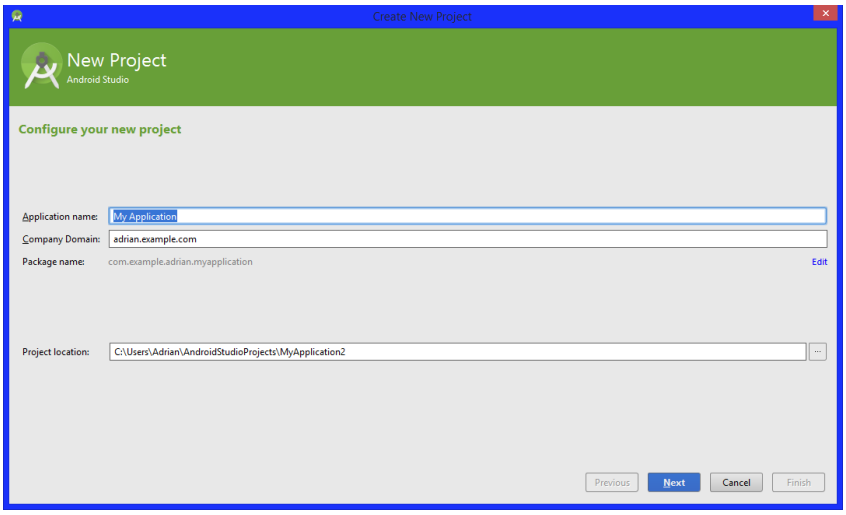


*Ilustración 51. Generación nuevo proyecto.*

El siguiente paso es muy importante, ya que es donde se indica para qué dispositivo o dispositivos va a estar orientada la aplicación a desarrollar. En este caso, la aplicación está pensada exclusivamente para su uso en un televisor, por lo que se marca dicha casilla. Como se comentó anteriormente, el API mínimo para el desarrollo en Android TV es el 21, y dicha opción se muestra automáticamente al seleccionar la TV como dispositivo como se puede apreciar en la ilustración 53.

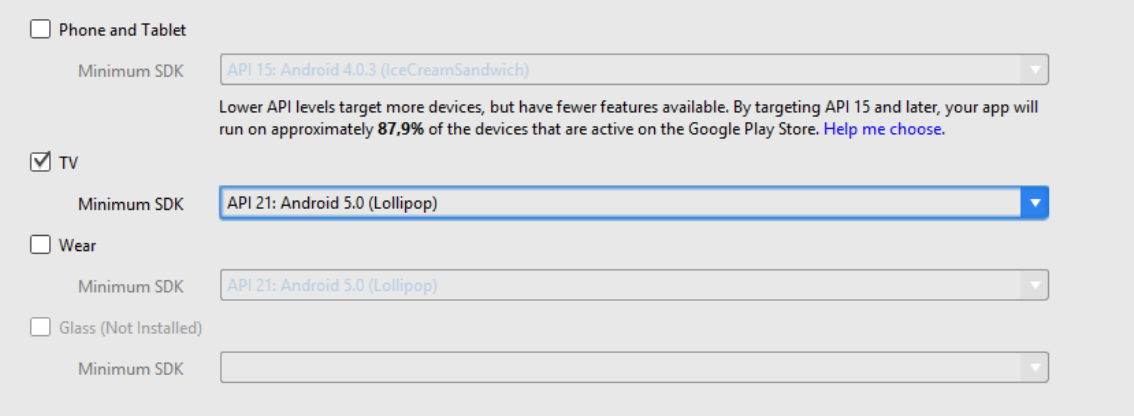


*Ilustración 52. Slección de dispositivo y API mínimo.*

Una vez seleccionado el dispositivo y el API mínimo, el programa da la opción de generar una serie de clases, de manera automática, a modo de ejemplo como aplicación de Android TV. De hecho, es el código base desde el cual se desarrolla la aplicación final de este trabajo.

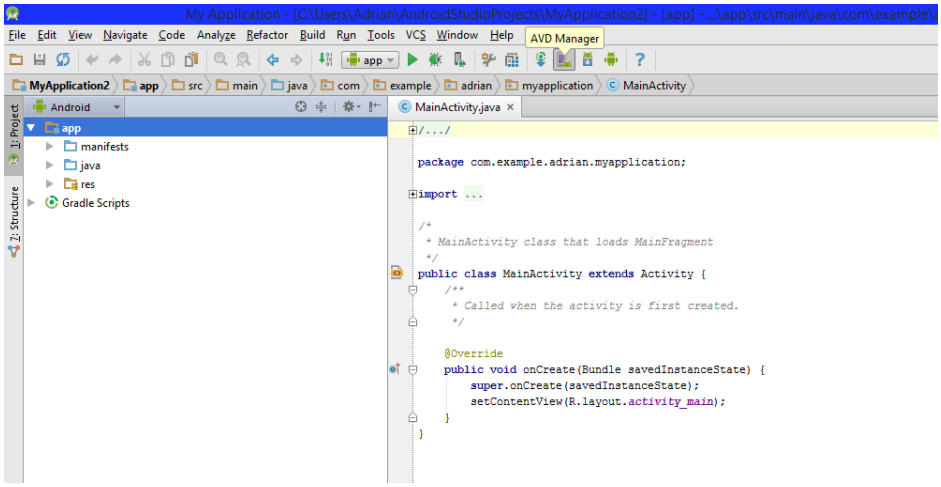


*Ilustración 53. Proyecto Android Tv generado correctamente.*

Una vez llegado a este punto, el entorno está correctamente configurado para el desarrollo en Android TV y el código de ejemplo incluye una *activity* con el launcher *Leanback* para el desarrollo en televisores y las librerías *Leanback* importadas en dicho proyecto de prueba.

Por tanto, lo único que falta es configurar un emulador adecuado para el testeo de aplicaciones Android TV. Para ello, el entorno proporciona una serie de dispositivos virtuales, algunos de ellos se corresponden con dispositivos reales, para la realización de pruebas.

En la figura 54 se muestran los diferentes dispositivos virtuales, en este caso para TV. A modo de ejemplo se selecciona un televisor Android TV con resolución HD, de 55 pulgada.

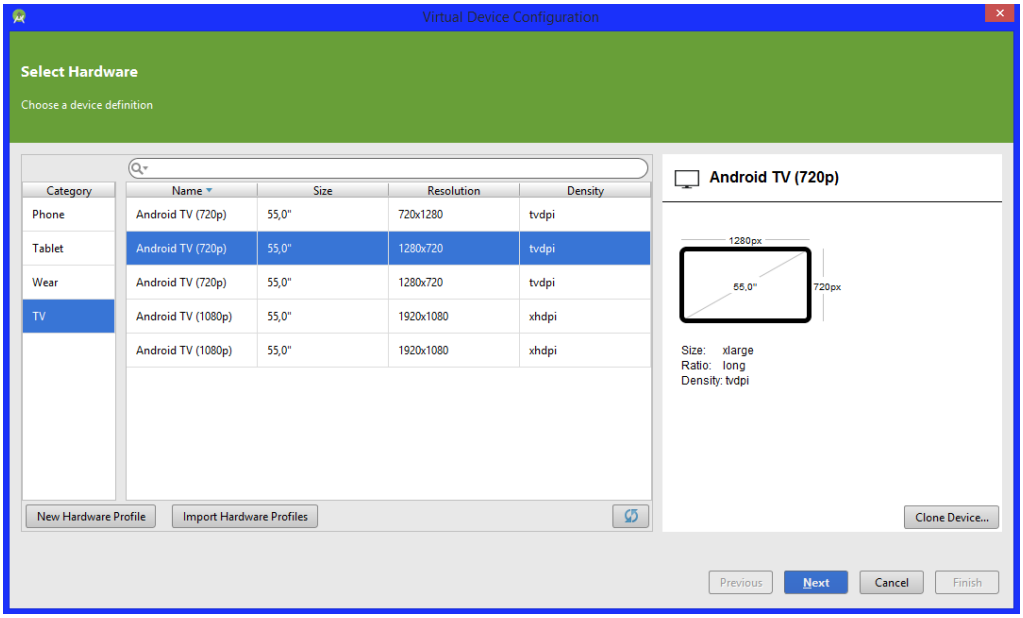


*Ilustración 54. Dispositivo virtual Android TV para el emulador.*

Por último, el entorno permite establecer un nombre diferente a dicho dispositivo, seleccionar si el emulador se quiere que sea *ARM* o *Intel* y un par de opciones para mejorar su rendimiento. La primera, *Use Host GPU*, para que el emulador utilice la GPU del propio ordenador y la segunda, *Start a snapshot for faster startup,* para que se guarde en memoria RAM una instancia del emulador una vez que haya arrancado y así incrementar la velocidad de sucesivos arranques. Sólo se puede elegir una de las dos opciones. Todo ello se puede apreciar en la ilustración 55.

Se pueden modificar algunos parámetros más avanzados en la opción de *Show Advanced Settings*, como la cantidad de memoria RAM que puede utilizar el emulador o configurar la simulación de una tarjeta de memoria.

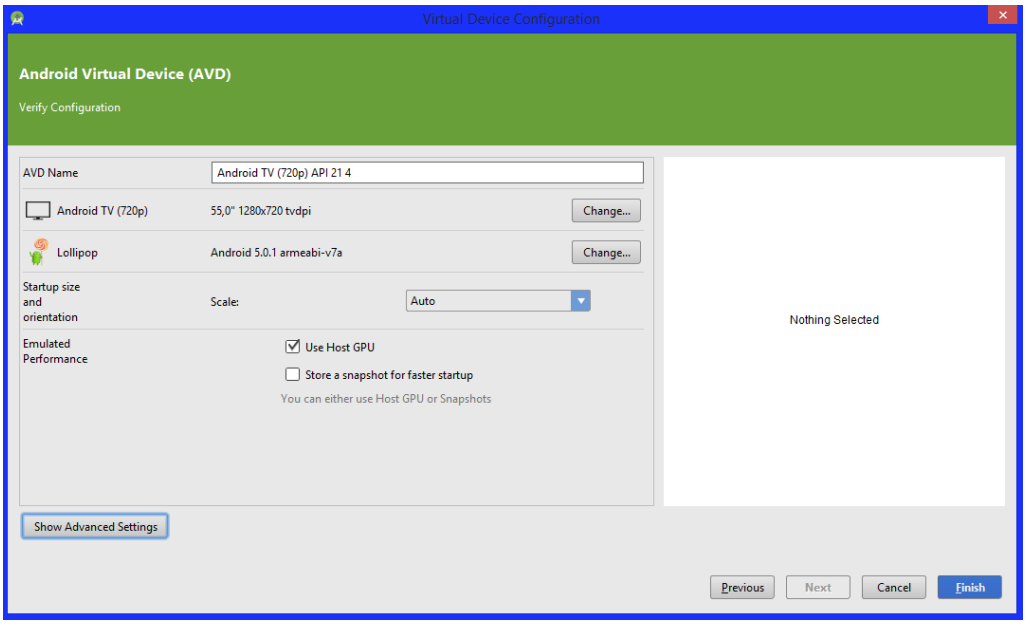


*Ilustración 55. Configuración del emulador.*